TURING 图灵程序设计丛书

Peachpit Press

累积销量超过100万册



HTML and CSS Visual QuickStart Guide Eighth Edition

HTML5与CSS3基础教程

（第8版）

[美] Elizabeth Castro 著
Bruce Hyslop
望以文 译

- 全球最畅销Web入门书最新版
- 配套网站提供海量精彩示例
- 逐步指导快速创建响应式网站

人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。



Elizabeth Castro

享誉世界的计算机畅销书作家，电子出版先行者，2010年就出版了epub电子书制作教程*EPUB Straight to the Point*。Castro擅长使用详实的步骤和精美的实例教大家快速实现具体效果，她的HTML与CSS系列教程自出版以来广受读者欢迎，成为学习前端开发的首选参考图书。有关Castro的更多信息，可访问其个人主页ElizabethCastro.com。



Bruce Hyslop

自1997年就开始从事网页开发，重点关注使用HTML、CSS和JavaScript进行网页开发和网站易用性的维护，并倡导最佳实践。Hyslop在加州大学洛杉矶分校进修部教授CSS课程，还著有*The HTML Pocket Guide*、*The Web Design Pocket Guide Boxed Set*。



望以文

毕业于中国人民大学，曾任百度前端工程师，现为网信金融产品经理。《HTML5与CSS3基础教程（第7版）》译者。热爱产品设计，维护微信公众号projojo，不定期向读者推荐新鲜有趣的互联网产品。微博@weakow。

TURING 图灵程序设计丛书

HTML and CSS
Visual QuickStart Guide, Eighth Edition

HTML5与CSS3基础教程
（第8版）

[美] Elizabeth Castro　Bruce Hyslop　著
望以文　译

人民邮电出版社
北　京

图书在版编目（CIP）数据

HTML5与CSS3基础教程 ：第8版 / (美) 卡斯特罗
(Castro,E.) ，(美) 希斯洛普 (Hyslop,B.) 著 ；望以
文译. -- 北京 ：人民邮电出版社，2014.5
（图灵程序设计丛书）
书名原文: HTML and CSS:visual quickstart guide
ISBN 978-7-115-35065-7

Ⅰ. ①H… Ⅱ. ①卡… ②希… ③望… Ⅲ. ①超文本
标记语言－程序设计－教材②网页制作工具－教材 Ⅳ.
①TP312②TP393.092

中国版本图书馆CIP数据核字(2014)第058722号

内 容 提 要

本书自第1版至今，一直是讲解HTML和CSS入门知识的经典畅销书，全面系统地阐述HTML5和CSS3基础知识以及实际运用技术，通过大量实例深入浅出地分析了网页制作的方方面面。最新第8版不仅介绍了文本、图像、链接、列表、表格、表单等网页元素，还介绍了如何为网页设计布局、添加动态效果等，另外还涉及调试和发布。本书提供了一个强大的配套网站，上面列出了书中的完整代码示例以及更多优秀实例及进阶参考资料，以供读者参考学习。

通过学习本书，零起点读者即可创建网站，而中高级水平的开发人员也可以快速了解HTML5新元素、CSS3的奇幻效果、响应式Web设计以及各种最佳实践。

◆ 著　[美] Elizabeth Castro　Bruce Hyslop
译　望以文
责任编辑　刘美英
责任印制　焦志炜
◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷
◆ 开本：800×1000　1/16
印张：27.25　彩插：4
字数：650千字　2014年5月第1版
印数：1－5 000册　2014年5月北京第1次印刷
著作权合同登记号　图字：01-2013-6737号

定价：69.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

版 权 声 明

译　者　序

HTML和CSS是万维网的根基。几年前，遵循Web标准之风盛行，网页结构与表现分离成为共识，网站开发人员普遍开始采用渐进增强的最佳实践，业内对HTML和CSS的认知度和重视度都得到了一次跨越式的提升。近年来，随着HTML5和CSS3技术的快速发展，HTML和CSS被赋予更强的功能性，它们的地位又到了前所未有的高度。

HTML和CSS的发展与变化，主要体现在两个方面。一是编写代码的方式有变化，例如，现在的智能手机普遍采用像素密度极高的Retina显示屏，如果还用过去的方式在网页中插入图像，这些图像在手机上就会显得不够锐利和清晰，因此我们有必要改变编写img标签的方式（参见本书5.8节）。二是新的技术取代了传统手段，例如，过去为了对标题等特殊文本使用不一样的字体，产生了图像替换文本的方法，现在，随着Web字体技术的成熟，人们可以使用更为简单、优雅的方式实现上述效果（参见本书第13章）。

过去的经验告诉我们，对于Web开发初学者来说，从一开始就学习最新技术和最佳实践，既可以节省时间，也有利于培养良好的编码习惯（因为避免了那些过时的编码方式和陈旧的技术手段的干扰）。

本书是HTML与CSS经典入门教程，目前已更新至第8版（第1版出版于18年前的万维网兴起之初）。跟过去的所有版本一样，本次新版也力图反映最新的编码实践与技术。相较于上一版，这一版几乎对全书的每个章节都有更新——删除了一些过时的做法，增加了不少新的提示与补充材料，大部分代码示例都进行了重写（相应的浏览器截图也都做了更新）。第8版不仅增加了一些新的知识板块（例如创建和使用为Retina显示屏准备的图像，通过使用可伸缩图像、弹性布局和媒体查询实现响应式网站等），还重新梳理了一些重点知识（例如对CSS层叠规则的讲解就重新整理了特殊性、顺序和重要性的关系）。

这套教程的经典之处是独特的分步讲解形式，贯穿全书的统一示例，以及通过提示和补充材料给出的丰富的扩展学习资源，因此对初学者而言，这套图书历来是学习效率的保证。新版不仅保留了这些经典的做法，还紧跟技术发展对内容做了全面更新，如果你是Web开发与设计的初学者，没有理由不将此书作为唯一的HTML与CSS入门书。

最后，借此机会，感谢图灵公司对我的信任与支持，感谢刘美英编辑，感谢她付出艰辛的努力，确保了本书按时出版。

前　言

无论你是刚开始涉足网站制作，还是曾经做过网站、而今又想让自己的技能与时俱进，都会对这个领域振奋人心的进展感同身受。

在过去的几年里，为网页编写代码和添加样式的方式、浏览网页所用的浏览器以及使用浏览器的设备都发生了明显的变化。以前，我们只能通过台式机或笔记本浏览万维网，而如今我们可以通过很多设备访问万维网：手机、平板电脑，当然也包括笔记本、台式机，以及很多别的设备。

这是意料之中的事，因为万维网始终秉持消除边界的宗旨——无论是在城市还是在乡村，通过任何能访问万维网的设备，任何人都可以自由地分享和获取信息。总之，万维网的宗旨是普适性。万维网的覆盖范围不断扩大，过去，通信技术还未曾普及农村，而现在却已经实现了。

更伟大的是，万维网是属于每一个人的，任何人都可以自由地创建和发布网站。本书将指导读者创建和发布网站，是没有任何HTML和CSS知识的建站初学者的理想教程。书中内容结构清晰，浅显易懂，将一步一步地教会你如何创建网页。总之，本书是开发人员手头案边的必备指南。若想了解某个主题的更多信息，可以通过查找目录直接跳到相关页面。

HTML 和 CSS 简介

万维网成功的根基，是一种基于文本的标记语言，它简单易学，并且能被任何带有基本Web浏览器的设备识读，它就是HTML。每个网页都或多或少会用一点儿HTML，否则也就不能称为网页了。

随着不断深入细致地学习，你会了解到，HTML用于定义内容的含义，而CSS用于定义内容和网页如何显示。HTML页面和CSS文件（**样式表**，stylesheet）都是文本文件，因此很容易编辑。后面的“如何学习本书”会向你展示一些HTML和CSS的代码片段。

从第1章起，我们开始学习基本的HTML页面；从第7章起学习用CSS定义页面样式。关于本书各章内容概述及主题汇总，请参见“本书涉及内容”。

HTML一词通常泛指这门语言本身，而HTML5则指代这一特定版本的HTML，比如讨论HTML早期版本所没有的HTML5新特性时。这样的规则也适用于CSS（泛指）和CSS3（特指CSS3）。

1. HTML与HTML5

了解一些有关HTML起源的基础知识对于理解HTML5很有帮助。

HTML诞生于20世纪90年代初，用于详细规定少量构建网页的元素。这些元素中的大多数都用于描述网页内容，如标题、段落、列表、指向其他网页的链接等。随着更多元素的引入以及对语法规则本身的调整，HTML这门语言的版本号也在更新。当前最新的版本便是HTML5。

HTML5是HTML早期版本的自然延续，它尽可能地满足了当前网站和未来网站的需求。它

从以前的版本中继承了大部分特性，这意味着，如果你在HTML5出现之前编写过HTML，那么你已经知道很多关于HTML5的知识了。这也意味着，HTML5的大部分内容都可以兼容新旧浏览器。向后兼容是HTML5的一项重要设计原则（参见www.w3.org/TR/html-design-principles/）。

HTML5还增加了大量新的功能。很多新功能都很简单，比如用于描述内容的辅助元素（如article、main、figure等）；还有一些用来协助创建强大Web应用程序的新功能则较为复杂。只有牢牢掌握了创建网页的技能，才能去学习HTML5更复杂的功能，这也是本书强调基础知识的原因。HTML5还引入了原生的音频和视频播放功能，书中也会讲到。

2. CSS与CSS3

CSS的第一个版本是在HTML诞生几年后才出现的，正式推出是1996年。同HTML5与其早期版本的关系一样，CSS3也是CSS早期版本的自然延续。

CSS3比CSS早期版本更为强大，它引入了大量的视觉效果，如阴影、圆角、渐变等。（关于本书涵盖这方面信息的详细情况，请参见“本书涉及内容”。）

Web 浏览器

无论是使用电脑（如图0.2.1所示）、手机，还是其他的设备，我们都是通过Web浏览器访问网站的。不过，你使用的浏览器可能与其他用户使用的浏览器不同。

Windows预装了微软公司的浏览器Internet Explorer，OS X则预装了苹果公司的浏览器Safari。不过，用户可以免费下载其他浏览器替换上面的两种浏览器，如Chrome（谷歌出品）、Firefox（Mozilla出品）、Opera（Opera出品）。以上讨论的都是用于桌面电脑的浏览器。

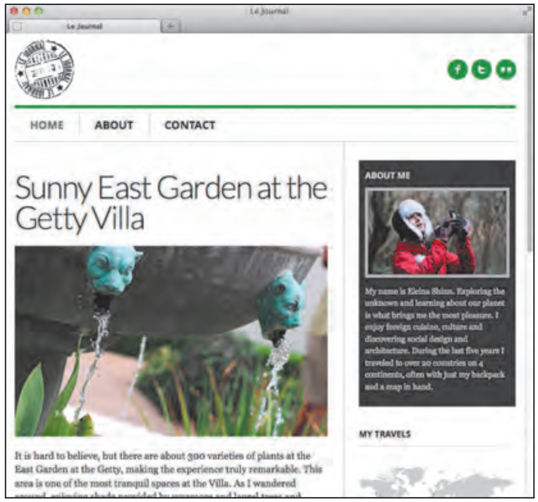


图0.2.1　Firefox的桌面版

至于移动设备，用户可以使用Safari的移动版（用于iPhone、iPad和iPod touch），各种Android默认浏览器——用于Android的Chrome、Firefox以及Opera Mini等。

本书会提到各种各样的浏览器。在大多数情况下，各个浏览器的最新版本对本书所讲的HTML和CSS特性的支持情况都是相似的。不过有时某个特性可能在一个或多个浏览器上不起作用（或表现不一致）。出现这些情况我会进行提示，并给出解决办法。这类情况主要发生在Internet Explorer 8上，这是目前开发人员还有必要关注的浏览器中最老的一个。（它的使用率仍在不断降低，因此2014年前后这种情况可能发生改变。）

20.2节会详细说明如何获取不同的浏览器，哪些浏览器对测试最重要，以及如何测试页面等问题。

浏览器的版本号

同HTML和CSS一样，浏览器也有版本号。版本号的数字越大，对应的浏览器就越新。

例如，Safari 7比Safari 6要新，而Safari 6则比Safari 5要新。Internet Explorer 10比Internet Explorer 9新。不过Internet Explorer 10并不比Safari 7新。

出现这种情况是因为微软、苹果以及其他浏览器厂商并不会相互协调各自浏览器的版本号，也不会协商新版浏览器的发布时间。Chrome和Firefox每隔六周就会更新一次，而其他浏览器大约要间隔一年才更新一次，因此Chrome和Firefox的版本号自然比其他浏览器的版本号要大得多。

很明显，无论是哪家厂商发布浏览器，新版本的浏览器对HTML和CSS特性的支持情况（当然也包括对其他特性的支持情况）都要好过先前的版本。

Web 标准与规范

你可能在想，是谁第一个创造了HTML和CSS，又是谁在持续地发展它们。由万维网和HTML的发明者Tim Berners-Lee主持的万维网联盟（W3C）便是负责带领Web标准发展的组织。

W3C发布的说明Web标准的文档称为**规范**（specification，缩写为spec）。这些规范对HTML、CSS等语言的参数进行了定义。也就是说，规范对语言规则进行了标准化。要了解W3C的活动，请访问www.w3.org（如图0.3.1所示）。

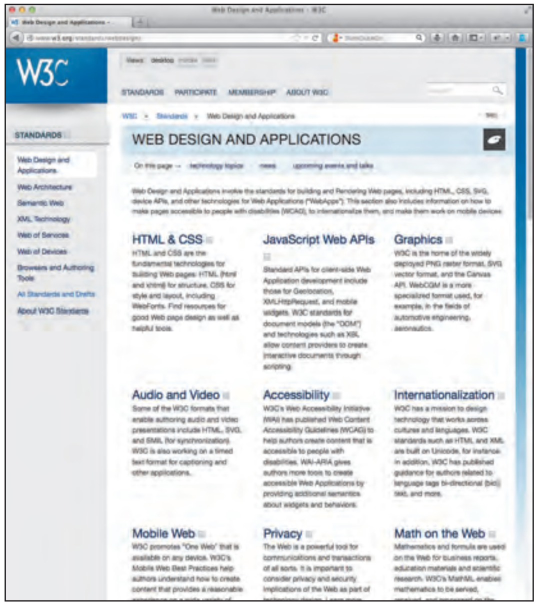


图0.3.1　W3C的网站是业内Web标准规范的主要信息来源

W3C与WHATWG

由于种种原因，另一个组织，Web超文本应用技术工作组（WHATWG），在负责开发HTML5规范的大量内容。W3C将WHATWG的工作纳入了其正在开发的规范的正式版本之中。可以通过www.whatwg.org访问WHATWG。

如果你想仔细查看这些规范（推荐你这样做），可访问下面列出的这些规范最新版本的网址。

- HTML5（W3C）
  http://www.w3.org/TR/html5/
- HTML5.1（W3C）
  http://www.w3.org/TR/html51/
- 开发中的HTML标准（WHATWG）
  http://www.whatwg.org/specs/web-apps/current-work/multipage/

开发中的HTML标准包含了一些比HTML5.1规范更新的特性，这些特性尚处于开发状态（很多部分还在不断变动）。最终，这些内容会融入HTML5.1规范。

CSS的规范太多了，无法全部列在这里，不过可以通过这个网址找到CSS规范的列表：http://www.w3.org/standards/techs/css#w3c_all。

HTML4和HTML5的区别

如果你之前接触过HTML4，并想了解它与HTML5的区别，可以查看W3C为此创建的文档（http://www.w3.org/TR/html5-diff）。

本书会提到很多HTML4和HTML5的区别。这些内容对HTML的初学者来说并不重要，因为实际上现在大家都会使用HTML5。不过，如果你想进一步研究，这些内容也很有趣。

利用已经完成的标准，我们可以根据一套成形的规则创建网页，而浏览器也能根据这套规则显示这些页面。（整体上，浏览器对标准的支持是良好的。Internet Explorer的早期版本，尤其是Internet Explorer 8，则存在一些问题。）

经过几个阶段的发展，规范最终会被确定下来，成为**推荐标准**（Recommendation，www.w3.org/2005/10/Process-20051014/tr）。

尽管一部分HTML5和CSS3规范仍未最终敲定，但这并不意味着你不能使用这些规范。标准化的过程需要经历一些时间（准确地说需要数年）。浏览器会在规范成为推荐标准之前很久便实现其功能，这也是规范发展过程的反映。因此，浏览器已经包括了大量HTML5和CSS3的功能，尽管它们还没有成为推荐标准。

总之，本书涉及的功能都是规范中相对明确规定的，因此它们在推荐标准推出之前发生改变的可能性极小。很多HTML5和CSS3功能已经实际使用一段时间了，开发者完全可以放心使用。

渐进增强：一种最佳实践

我在前言的开头处就讲到了万维网的普适性——万维网上的信息应该能被所有人访问。**渐进增强**（progressive enhancement）的理念能帮助你构建具有普适性的网站。这不是一门语言，而是一种建站方法，它由Steve Champeon于2003年提出（http://en.wikipedia.org/wiki/Progressive_enhancement）。

这个想法很简单，但也很强大：开始用所有人都能访问的HTML内容和行为构建网站（如图0.4.1所示），再用CSS加入你的设计（如图0.4.2所示），最后用JavaScript（一种编程语言）添加额外的行为。这些组件都是分离的，但可以同时发挥作用。

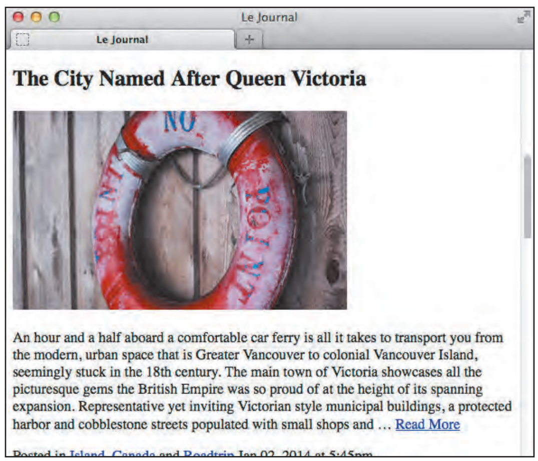


图0.4.1 一个基本的HTML页面，未对其应用任何自定义的CSS。通常，只有比较旧的浏览器才会以这种方式显示。这个页面可能并不是很好看，但信息都是可访问的，这一点非常重要



图0.4.2　同一个页面在支持CSS的浏览器中的显示效果。信息是相同的，只是呈现的方式不同（将页面向下滚动，右侧内容在图0.4.1中也是可见的）

更多例子

遵循渐进增强的原则，可以让网站的布局适应不同设备的屏幕尺寸和浏览器功能。这样，网页在移动设备、桌面电脑甚至更高级的设备上看起来都很不错。如果你对这个主题很感兴趣，可以先阅读第12章。

第14章介绍了如何让旧的浏览器显示简化的设计而现代浏览器显示包含CSS3效果的增强版本。

在本书的其他部分，你也会学到用于构建渐进增强网页的其他技术。

这样做的结果就是，那些只能访问基本页面的设备和浏览器得到的是简化的、默认的体验（如图0.4.1所示）。即便是20多年前万维网刚诞生时的浏览器也能显示这个页面，因此最早配备Web浏览器的手机也能显示它。此外，屏幕阅读器（可以为视障访问者读出网页内容的软件）也能轻松地处理这个页面。

同时，那些能够浏览更健壮的网站的设备和浏览器将看到增强的版本（如图0.4.2所示）。还有一些浏览器的处理能力介于二者之间，它们也能显示这个页面。不必要求网站对所有人来说体验都是一样的，关键是网站的内容是可访问的。

本质上，渐进增强背后的涵义是共赢。

目标读者

本书假定读者没有任何创建网站的知识。因此，从这一点上说，本书适合零起点初学者。你将从最基础的HTML和CSS知识学起。在这个过程中，你也会学到HTML5和CSS3的新特性，尤其是那些设计师和开发者在日常工作中经常用到的特性。

不过，即使你对HTML和CSS很熟悉，依然可以从本书获益，阅读本书可以快速了解HTML5的新元素、一些CSS3效果、响应式Web设计以及各种最佳实践。

1. 本书涉及内容

全书各章是按照如下方式进行组织的。

- ❑ 第1章～第6章及第15章～第18章讲解创建HTML页面的原则和我们需要使用的HTML元素，并清楚地说明什么时候该使用这些元素，以及如何使用。
- ❑ 第7章～第14章讲解CSS，从创建第一条样式规则开始，最终讲到运用CSS3增强视觉效果。
- ❑ 第19章演示如何向页面添加预先写好的JavaScript。
- ❑ 第20章讲解在将页面放到万维网上之前如何对其进行测试和调试。
- ❑ 第21章讲解如何保护自己的域名，以及如何把网站放到万维网上让其他所有人都能看到。

本书涵盖以下主题。

- ❑ 创建、保存、编辑HTML和CSS文件。
- ❑ 编写语义化HTML的含义及其重要性。
- ❑ 如何把页面的HTML内容、CSS表现和JavaScript行为分离（这是渐进增强的关键所在）。

- ❑ 通过一种有含义的方式，使用那些存在多年的HTML元素和新增的HTML5元素对内容进行结构化。
- ❑ 从一个网页链接到另一个网页，或从页面的一部分链接到另一部分。
- ❑ 在页面中添加图像，并针对万维网对其进行优化。这其中还包括了创建适应苹果的Retina显示屏及其他高分辨率显示屏的图像。
- ❑ 使用ARIA（无障碍富因特网应用）地标角色和其他好的编码实践提升网站的可访问性。
- ❑ 为文字添加样式（大小、颜色、粗体、斜体等），添加背景颜色和背景图像。
- ❑ 实现多栏布局。
- ❑ 构建响应式网页。响应式网页可以根据访问者的屏幕大小收缩或放大布局，从而按照开发者希望的方式适应不同的设备。这样做的结果是页面在手机、平板电脑、笔记本电脑、台式机以及其他可访问万维网的设备上看起来效果都很不错。
- ❑ 使用`@font-face`为页面添加自定义Web字体，使用Font Squirrel和Google Fonts提供的字体服务。
- ❑ 应用CSS3效果，如透明度、背景alpha透明、渐变、圆角、外阴影、内阴影、文字阴影以及多背景图像。
- ❑ 使用CSS生成内容和sprite以减少页面所需图像文件的数量，让页面加载得更快。
- ❑ 创建表单来获取访问者输入的内容，包括使用一些HTML5中新增的输入类型。
- ❑ 使用HTML5的`audio`和`video`元素向页面插入媒体，对旧的浏览器使用备用的Flash播放器。
- ❑ 还有很多主题。

这些主题都配有很多代码示例，说明如何基于业内最佳实践来实现HTML5和CSS3的各个特性。

2. 本书未涉及内容

近年来，HTML和CSS领域的发展很迅猛，我们不得不舍弃一些主题。除了几处例外，我们坚持把那些极少用到、仍可能变化、缺乏广泛浏览器支持或者需要JavaScript知识的主题，以及一些高级主题排除在本书范围以外。

本书没有涉及下列主题。

- ❑ HTML5的`details`、`summary`、`menu`、`command`、`output`和`keygen`元素。W3C将其中的几个元素列入了2014年HTML5最终定稿时可能不包含的功能，应该尽量避免使用这里提到的元素。
- ❑ HTML5的canvas，这个元素允许我们使用JavaScript绘制图像（甚至创建游戏）。此外，还有可缩放矢量图形（SVG）。第17章对这两个主题进行了简要说明，还包含了一些涉及相关详细信息的网址。
- ❑ 需要JavaScript知识或者与HTML5新增的语义化元素没有直接关联的HTML5 API和其他高级特性。
- ❑ CSS3变换、动画和过渡效果。要了解更多信息，参见www.htmlcssvqs.com/resources/。
- ❑ CSS3中新的布局模型，如FlexBox、Grid等。这些技术致力于改变我们对页面进行布局的方法（前提是规范进一步发展，浏览器的支持程度更高一些）。参见Zoe Mickley Gillenwater的演示文档（www.slideshare.net/zomigi/css3-layout）和Peter Gasston的文章（www.netmagazine.com/features/pros-guide-css-layouts）。

如何使用本书

本书几乎所有章节都有实战代码示例（如图0.6.1和图0.6.2所示）。通常情况下，随后还会

展示这些代码对应的页面在浏览器中显示的屏幕截图（如图0.6.3所示）。

```
...
<body>
<header class="masthead" role="banner">
    ...
        <nav role="navigation">
            <ul class="nav-main">
                <li><a href="/" class="current-
                → page">Home</a></li>
                <li><a href="/about/">About</a>
                → </li>
                <li><a href="/contact/">Contact
                → </a></li>
            </ul>
        </nav>
    ...
</header>
...
</body>
</html>
```

图0.6.1　本书大部分章节都给出了一些HTML代码片段，且会突出显示跟相应章节内容对应的部分。省略号（...）表示出于简洁而省略的额外代码或内容。通常，可在其他代码示例中看到省略的部分

```
body {
    font-family: Georgia, "Times New Roman",
    → serif;
}

/* 站点导航 */
.nav-main {
    list-style: none;
    padding: .45em 0 .5em;
}

.nav-main li {
    border-left: 1px solid #c8c8c8;
}

.nav-main a {
    color: #292929;
    font-size: 1.125em;
    font-weight: bold;
}
```

图0.6.2　如果示例有CSS代码，会将它们单独显示出来，对应内容也会突出显示

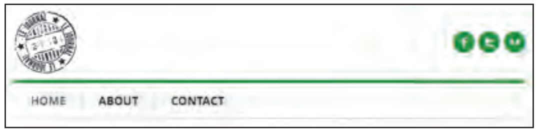


图0.6.3　用于演示代码如何影响页面的一种或多种浏览器的屏幕截图

大多数屏幕截图用的都是当下最新版本的Firefox。不过，这并不是说与其他浏览器相比，我们更推荐Firefox。本书的代码示例在最新版本的Chrome、Internet Explorer、Opera和Safari中看起来都是相似的。

代码和屏幕截图下面都有对相应HTML元素和CSS属性的描述，以便于对示例背景进行说明，并加强你对它们的理解。

在很多情况下，你会发现只要理解说明和代码示例就可以使用HTML和CSS特性了。但如果你需要关于如何使用这些特性的明确指导，也可以阅读书中列出的详细步骤。

最后，大部分章节都有一些提示，为读者提供额外的用法信息、最佳实践、对本书相关部分的引用、对其他资源的链接等。

本书约定

本书遵循以下约定。

- 需要开发人员输入值的占位符或代码以斜体显示。大多数占位符出现在步骤讲解中。例如，输入`padding: `*x*，这里的*x*是要添加的内边距。
- 需要准确输入的代码（即HTML和CSS代码）使用等宽字体（如`this font`）。
- 代码图中的箭头（→）代表延续上一行的代码（这一行代码由于篇幅原因而进行了折行），如图0.6.2所示。箭头并非代码的一部分，不需要输入。实际使用中应该连续输入这一行代码，而无需像印刷页面中这样折行。

- 对于首次出现的词，在定义时使用黑体显示。
- IE是对Internet Explorer的简称。例如，IE9与Internet Explorer 9同义。
- 对HTML5和CSS3最新特性支持较好的浏览器统称为**现代浏览器**（modern browser）。通常，“Web浏览器”一节中提到的浏览器的最新版本都属于现代浏览器（IE8除外）。
- 浏览器的版本号后面出现加号（+）指的是该版本及其之后的所有版本。例如，IE8+指的是Internet Explorer 8及其后续版本。

本书配套网站

本书的配套网站上有本书的目录、本书涉及的（以及其他一些不方便放入书中的）完整代码示例、引用资源的链接（以及一些额外资源的链接）、勘误表等。

下面是本书网站一些重要页面的URL。

- 主页：www.htmlcssvqs.com。
- 代码示例：www.htmlcssvqs.com/8ed/examples/。

读者可以直接在网页上查看这些代码，也可以把它们下载到自己的电脑里。读者可以获取书中涉及的所有HTML和CSS文件。

有时，我们会在代码里插入一些注释，解释对应代码的含义和用法。限于篇幅，我们对书中大量的代码示例进行了删减，但配套网站列出了完整的代码。

读者可随意使用这些代码，对它们进行修改使之满足自己项目的需要。

希望该网站对你有用。

致谢

撰写本书过程中最有意思的就是与很多优秀的人士合作。所有的人都很用心、专业、和善且富有幽默感，让著书成为乐事。没有他们的努力，就不会有这本书。

将最诚挚的感谢献给以下各位。

Nancy Aldrich-Ruenzel和Nancy Davis，感谢他们对我的信任。

Cliff Colby，感谢他的支持，是他建立了我们的团队，让一切工作井井有条。

Robyn Thomas，感谢他让我们富有激情地工作，对内容提出建议，严格要求细节，并不断给予我们鼓励。

Scout Festa，感谢她简化语言的能力以及对本书的严厉监督，从而确保了全书的语言统一、文字优美流畅。

Aubrey Taylor，感谢他的众多建议和技术支持。这些内容对读者来说都非常有用。

David Van Ness，感谢他细致的页面排版工作，他的工作让全书看起来相当漂亮。

Valerie Haynes Perry，感谢他创建了一个高效的索引，可供读者反复查阅。

Peachpit公司的市场、销售等工作人员，感谢他们的幕后工作让本书得以面市。

Natalia Ammon，感谢她为第11章和第12章以及其他一些位置设计的漂亮页面。她的作品地址为www.nataliaammon.com。

Zach Szukala，感谢他将Natalia推荐给我。

Scott Boms、Ian Devlin、Seth Lemoine、Erik Vorhes和Brian Warren，感谢他们对本书上一版的贡献。

Victor Gavenda，感谢他帮助我们获取必要的软件。

Dan Cederholm、Ethan Dunham、Paul Irish、Mark James、Randy Jensen、Julie Lamba、Fontfabric、Google和Namecheap，感谢他们允许使用一些屏幕截图和设计资源（视具体情况而定）。

C.R. Freer，感谢她魔法般的摄影。

我的家人和朋友，感谢他们的鼓励，以及在写作间隙的陪伴。感谢他们在我几个月夜以继日的写作过程中不离不弃。

Robert Reinhardt，和前几版中的致谢一样，感谢他引领我接触写作，感谢他建议我蓄起美髯。

波士顿棕熊队[①]，感谢他们在我写作间歇贡献了那么多激动人心的季后赛。

Web社区的人们，感谢他们分享技术知识，从而让他人获益（我在全书多处引用了他们的内容）。

读者朋友，感谢你们让我唤起自己当初学习HTML和CSS的记忆，让我得以按照最能让你们受益的方式讲解这些内容。感谢你们选择本书作为踏入万维网创作世界这一旅程的一部分。

① 波士顿棕熊队是美国波士顿的一支冰球队。——译者注

最后，特别感谢Elizabeth Castro。她在20世纪90年代打造了这个品牌。多年来，本书一版再版，为无数读者提供了帮助。万维网赋予我太多珍贵的东西，因此我尤其珍惜使用这一书名让读者学习HTML和CSS的机会。

Bruce



图 5.1.2　标识和其他颜色较少的图像通常保存为 PNG-8 格式（也可以保存为 GIF 格式，不过相较之下 PNG 更好一些）

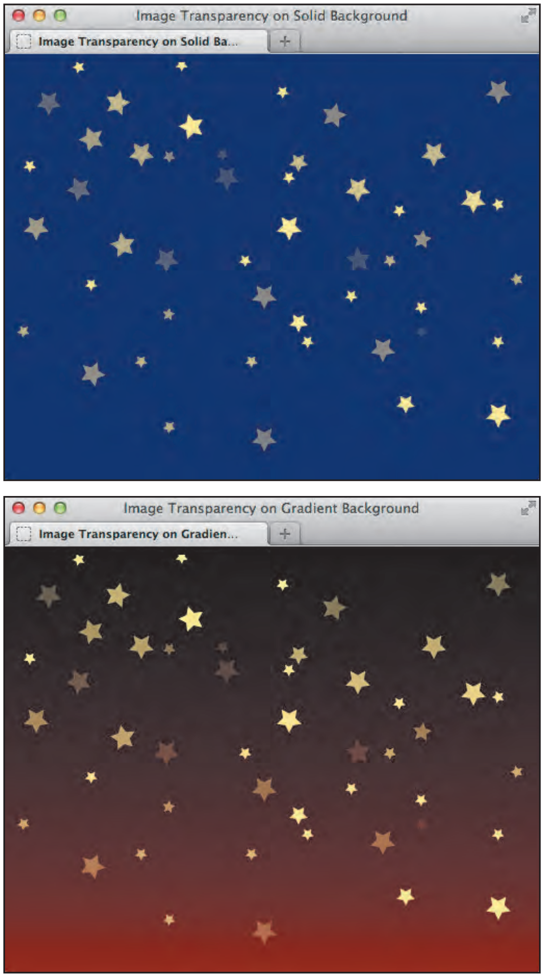


图 5.1.5　纯蓝色背景和从黑到红渐变色背景都不是星星 PNG 图的一部分，它们只是用 CSS 向页面的 body 添加的颜色。星星图具有 alpha 透明度，这样，无论背景是纯色、渐变色，还是另一张图像，就都能穿透，且看起来相当“干净”。JPEG 无法实现这一点，GIF 仅支持一种简单的透明度，效果并不太好（除非背景是纯色的）





图 5.4.4　这里的修改反映的是左下方的区块。如果保存它，其大小就是 62.39K，尽管在图 5.4.2 中它只有 19.44K，但现在它的边缘更锐利了，这种质量上的提升是值得的。这里还修改了其他两个区块的设置。PNG-8（右上）的压缩让图像的像素化程度加深了，而且文件大小（70.56K）也比 JPEG 要大。PNG-24（右下）保持了很高的质量，但文件大小（224.3K）也大得多。显然，JPEG 是保存照片的最佳选择



图 5.4.5　这个图像（已被放大）有很多平铺的颜色及文字，它们需要保持锐利。可以看到，PNG-8 格式（左下）对图像的压缩是最好的，文件大小只有 5.5K。如果减少颜色的数量，其大小还将进一步减小。PNG-24 可以提供更多的颜色，使用这种格式的话，文件大小为 11.17K。使用最高质量的 JPEG 则为 19.76K。如果将 JPEG 的质量降为 50（这里没有显示），文件大小可比 PNG-8 更小一点，但其效果看上去却很可怕

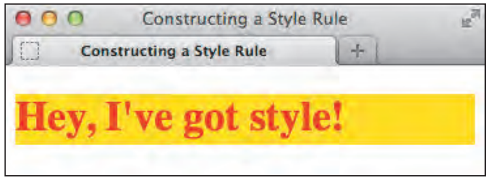


图 7.1.4　标题从浏览器默认的白色背景、黑色文字样式变为黄色背景、红色文字

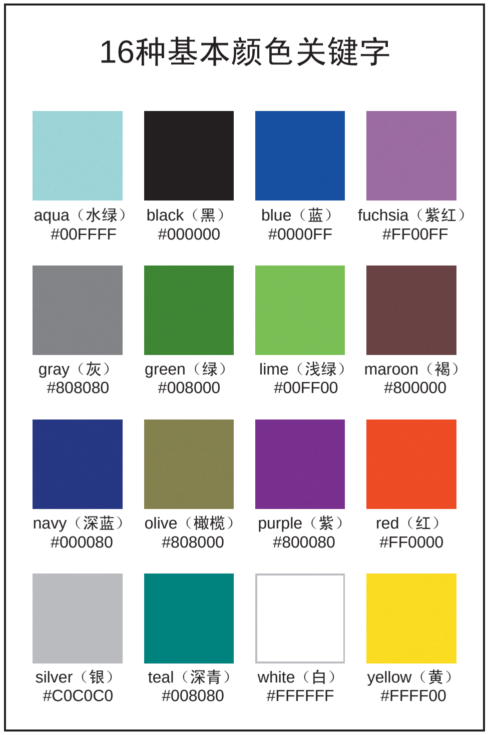


图 7.5.6　16 种基本颜色关键字，以及它们对应的十六进制数字值。CSS3 新增了 131 个关键字，不过通过十六进制、RGB 或 HSL 等格式可以定义多得多的颜色

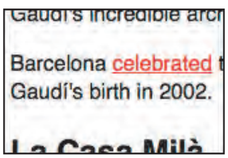


图 9.7.3　新的、未访问的链接显示为红色

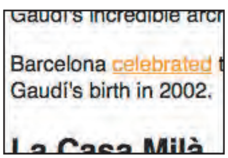


图 9.7.4　访问过的链接变为橙色

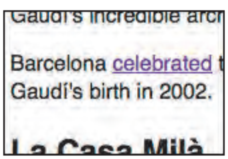


图 9.7.5　链接获得焦点（如通过 Tab 键）时显示为紫色

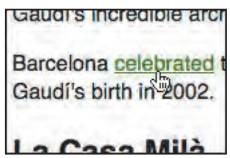


图 9.7.6　当访问者将鼠标指针停留在链接上时，它显示为绿色

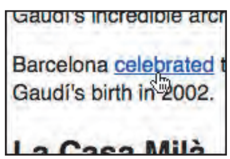


图 9.7.7　当访问者激活链接时，它变为蓝色

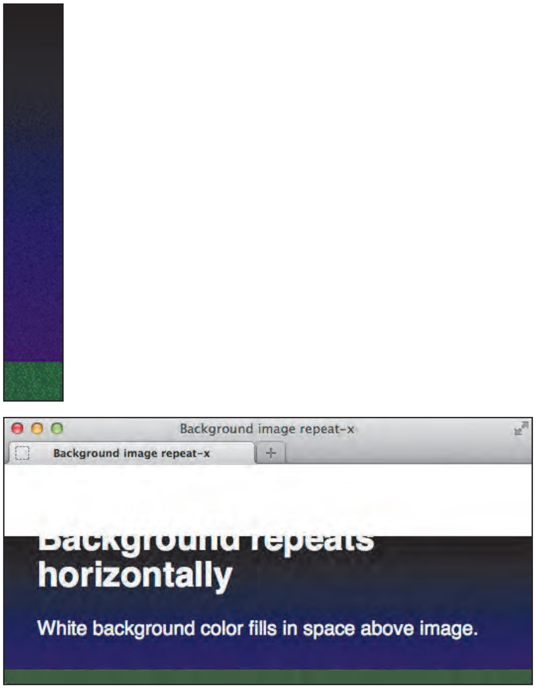


图 10.10.10　这张背景图像（上）由黑色向蓝色再向绿色渐变，很适合水平重复。不过它的高度不足以填充整个页面，在不设置背景色的情况下显得很难看。那一行文字其实并没有被切割，看不见它的上半部分是因为它的颜色与默认的背景色一样，都是白色

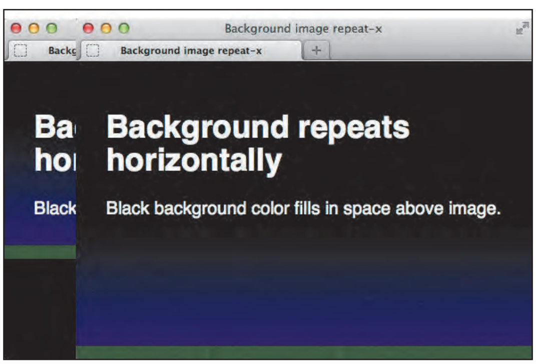


图 10.10.12　黑色（#000）背景色与图像顶端的黑色完美地融合在一起。现在，你可以看到那些文本了！这张图是由两个浏览器窗口截图合成的，这两个窗口的高度是一样的。如果没有为 html 元素添加图 10.10.11 中的规则，效果就如左图所示，背景图像的底部位于文字下面一点的位置（因为页面在那个位置结束了），因此黑色背景色会显示在它的下面

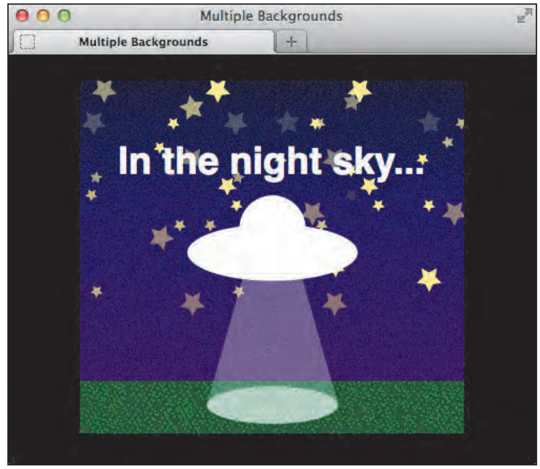


图 14.6.3　支持多重背景的浏览器显示示例的样子。其中，图像是分层次相互重叠在一起的，用逗号分隔的列表中的第一个图像位于顶部

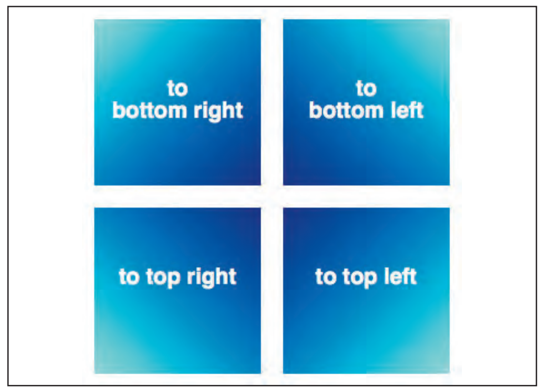


图 14.7.4　有两种指定渐变方向角度的方式。第一种方式就是这里所示的，使用关键字指定渐变方向需要旋转的角度。渐变会从跟指定角相对的角（对角线另一端的角）开始



图 14.7.5　使用关键字（参见图 14.7.4）只能创建沿对角线方向的渐变。第二种方式是指定渐变角度的度数，如 90deg（90 度）。数值代表的是圆周上的点的位置：0 代表最顶端的点，90 代表最左边的点，180 代表最底端的点，270 代表最右边的点。你列出的值决定的是渐变结束的点的位置。因此，0deg 等价于 `to top`，90deg 等价于 `to right`，以此类推



图 14.7.6　包含额外的可选参数的径向渐变。最简单的例子是默认的样式规则，它使用的参数与线性渐变是一样的。在这个例子中，渐变的原点是元素的中心。可以使用 at top 这样的关键字指定中心的位置。跟往常一样，可以在径向渐变声明之前为较旧的浏览器提供一个背景的定义



图 14.7.7　还可以控制渐变的尺寸。这里展示了几个例子

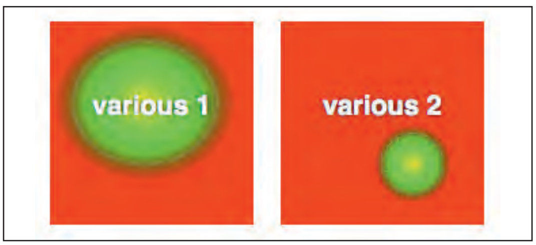


图 14.7.8　这些例子结合了几个参数。它们都有三个颜色。位于 at 后面的值指定的是渐变的中心坐标。在第二个例子中，30px 30px 标记了渐变的尺寸。在第一个例子中，closest-side 决定了渐变的尺寸，它告诉浏览器从中心（指定为 at 70px 60px）向包含该渐变的区域的最近的一边伸展

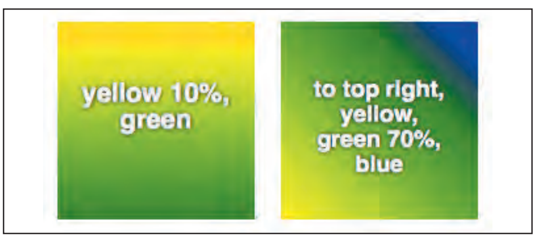


图 14.7.9　可以使用百分数指定一个以上的颜色停止位置

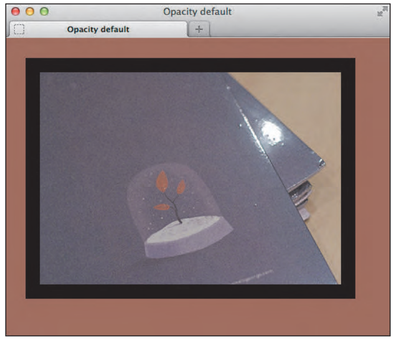


图 14.8.2　这是图 14.8.1 中 div 的默认样式。同其他元素一样，浏览器为其设置的默认样式为 opacity: 1;（不需要在样式表中指定）。该 div 有一个黑色的背景，由于该元素有内边距，因此背景会显示出来。body 的背景为褐色

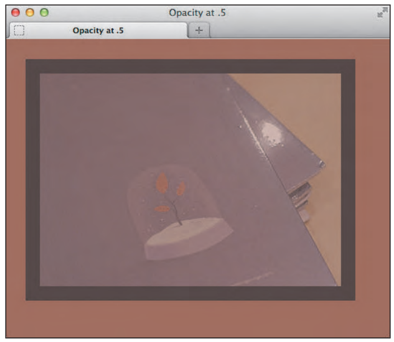


图 14.8.4　这是示例中 div 元素的 opacity 设为 0.5（即不透明度为 50%）时的样子。注意 div 元素的黑色背景现在呈现为深褐色，图像也变成半透明的，且带有一些褐色的色彩。如果将某元素的 opacity 设为小于 1 的值，包含它的周边元素就会透出来。在这个例子中，透出来的是 body 的褐色背景。如果将 body 的背景设为红色，效果会更明显（参见图 14.8.5）

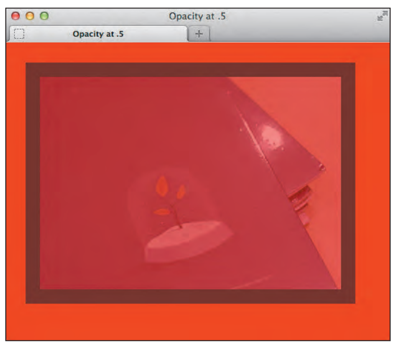


图 14.8.5　示例中 div 仍设为 opacity: .5;，但 body 改为 background-color: red;。通透的程度取决于不透明度。如果将不透明度设为 0.35，就会透出更多的红色；如果设为 0.8，则透出的红色就更少

目　　录

第 1 章

网页的构造块

1

本章内容

- HTML 思想
- 基本的 HTML 页面
- 标记：元素、属性、值及其他
- 网页的文本内容
- 链接、图像和其他非文本内容
- 文件名和文件夹名
- URL
- HTML：有含义的标记
- 浏览器对网页的默认显示效果
- 要点回顾

尽管网页变得越来越复杂，但是其底层结构依然相当简单。正如前言中提到的，创建网页是离不开 HTML 的。你即将了解到，HTML 包含网页内容并说明这些内容的意义，Web 浏览器则会将 HTML 包着的内容呈现给用户。

一个网页主要包括以下三个部分。

- **文本内容**（text content）：在页面上让访问者了解页面内容的纯文字，比如关于业务、产品、家庭度假的内容，以及页面关注的其他任何内容。
- **对其他文件的引用**（references to other files）：我们使用这些引用来加载图像、音频、视频文件，以及样式表（控制页面的显示效果）和 JavaScript 文件（为页面添加行为）。这些引用还可以指向其他的 HTML 页面和资源。
- **标记**（markup）：对文本内容进行描述并确保引用正确地工作。（HTML 一词中的字母 M 就代表标记。）

此外，在每个 HTML 页面的开头都有一些主要用于浏览器和搜索引擎（如 Bing、Duck Duck Go、Google、Yahoo 等）的信息。浏览器不会将这些信息呈现给访问者。

需要注意的是，网页的这些成分都仅由文本构成。这意味着网页可以保存为纯文本格式，可以在任何平台（无论是台式机、手机、平板电脑还是其他平台）上用任何浏览器查看。这个特性也确保了很容易创建 HTML 页面。

本章会带你创建一个基本的 HTML 页面，讲解 HTML 基础（包括上述三种主要成分），并讨论一些最佳实践。

注意　正如在前言中提到的，本书使用 HTML 泛指这门语言本身。如果需要突出 HTML 某一版本独有的特殊属性，则使用它们各自的名称。例如，“HTML5 引入了一些新的元素，并重新定义或删除了 HTML 4 和 XHTML 1.0 中的某些元素。”

1.1　HTML 思想

设想这样一种场景：你在厨房里，一只手上拿着便利贴，上面的每一页都写着一个

单词。有的贴纸上写的是“汤”，有的写着“燕麦片”、“盘子”、“酱汁”等。

你打开一个橱柜，为里面的每一样物品都贴上最能描述它的便利贴。一只黄色的装燕麦片的盒子贴上了写有“燕麦片”的贴纸。还有一个红色的盒子也是装燕麦片的，也为它贴上写有“燕麦片”的贴纸。其他的物品也采用类似的做法。

编写 HTML 与这个过程是很相似的。不同的是，编写 HTML 并非向食物和餐具上贴标签，而是为网页内容打上能够描述它们的标签。你无需自己创建标签名称，HTML 已经完成了这一工作，它有一套预先定义好的元素。p 元素用于段落，abbr 元素用于缩略词，li 元素用于列表项目。本书后面会进一步介绍这些元素，以及很多其他的元素。

注意，便利贴上的用词是“燕麦片”这样的，而不是“黄色的谷物盒子”、“红色的谷物盒子”。类似地，HTML 元素描述的是内容是什么，而非看起来是什么样。CSS（从第 7 章开始讲解）才控制内容的外观（如字体、颜色、阴影等）。因此，不管你最后让段落显示为绿色还是橙色，它们都是 p 元素，这才是 HTML 唯一关心的。

在学习本书和创建网页时，应该始终牢记这一思想。接下来即将讲到的基本网页就是这样做的。

1.2　基本的 HTML 页面

来看一个基本的 HTML 页面，大概了解一下本章及后面的内容。我们会对本节中的代码进行讲解。不过，即便你现在不能完全理解这些代码也不必担心，这只是让你初步了解 HTML，本书接下来的部分会对其进行详细说明。当然，随着你进一步阅读本书的后续内容，还会了解到更多的细节。此外，每个人的学习方法都不同，或许有的读者认为先阅读下一节，再返回来读这一节会更有效。

每个网页都由图 1.2.1 中所示的结构开始构建。这个 HTML 相当于一张白纸（参见图 1.2.2），因为访问者看到的内容位于**主体**（body）部分（即 <body> 和 </body> 之间的部分），而这一部分现在是空着的。我们马上就会给它填上内容。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Your page title</title>
</head>
<body>


</body>
</html>
```

图 1.2.1　每个网页都包含 DOCTYPE、html、head 和 body 元素，它们是网页的基础。在这个页面中，可以定制的内容包括两项，一是赋予 lang 属性的语言代码，二是 <title> 和 </title> 之间的文字

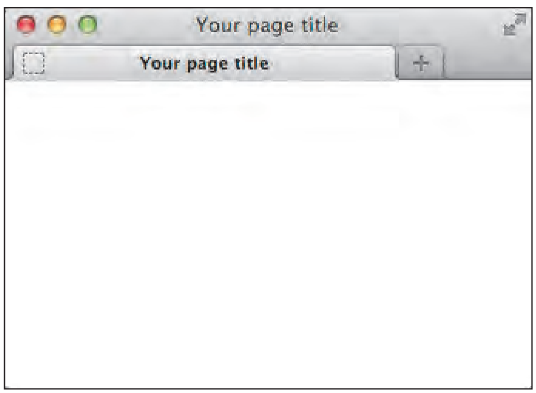


图 1.2.2　除非你是一个极简主义者，否则这不会是一个多么令人激动的页面

首先，简单说明一下（下一节会详细解释），HTML 使用 < 和 > 包围 HTML 标签。

开始标签（如 <head>）用于标记元素的开始，**结束标签**（如 </head>）用于标记元素的结束。有的元素没有结束标签，如 meta（参见图 1.2.1）。

1. 网页的顶部和头部

前面提到，页面内容位于主体部分，那么其他的代码有什么作用呢？实际上，<body> 开始标签以上的内容都是为浏览器和搜索引擎准备的。<!DOCTYPE html> 部分（称为 DOCTYPE）告诉浏览器这是一个 HTML5 页面。DOCTYPE 应该始终位于页面的第一行。

接下来是 html 元素，它包着页面的其余部分，即 <html lang="en"> 和 </html> 结束标签（表示页面的结尾）之间的内容。在这里可以为网页指定其他语言，这一内容将在 3.1 节作详细说明。

再接下来是文档的头部，即 <head> 和 </head> 之间的区域。<meta charset="utf-8" /> 会在 1.4 节进行讲解。主体前面的代码中，有一部分是用户可见的，即 <title> 和 </title> 之间的文本。这些文本会出现在浏览器标签页中，对于某些浏览器，这些文本还会出现在浏览器窗口的顶部，作为网页的标题（如图 1.2.2 所示）。此外，这些文本通常还是浏览器书签的默认名称，它们对搜索引擎来说也是非常重要的信息。

3.1 节和 3.2 节会对 DOCTYPE 和头部进行详细的讲解。

这就是网页的基础，并不太难，对吧？

2. 网页的主体：你的内容

接下来为页面添加一些内容（如图 1.2.3 所示）。图 1.2.4 显示了桌面浏览器呈现这段 HTML 的效果（如果添加 CSS，页面看起来会更漂亮）。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Blue Flax (Linum lewisii)</title>
</head>
<body>
<article>
    <h1>The Ephemeral Blue Flax</h1>

    <img src="blueflax.jpg" width="300" height="175" alt="Blue Flax" />

    <p>I am continually <em>amazed</em> at the beautiful, delicate <a href="http://en.wikipedia.
    ➝ org/wiki/Linum_lewisii" rel="external" title="Learn more about Blue Flax">Blue Flax</a>
    ➝ that somehow took hold in my garden. They are awash in color every morning, yet not a single
    ➝ flower remains by the afternoon. They are the very definition of ephemeral.</p>
</article>
</body>
</html>
```

图 1.2.3　这个页面包含了本章开头提到的三种成分：文本内容、对其他文件的引用（图像的 src 值和链接的 href 值）及标记。我突出显示了主体部分的标记，读者可以很方便的区分标记和页面的文本内容。同时，请注意，行与行之间通过回车符分开了，这并不是必需的，不影响页面的呈现效果

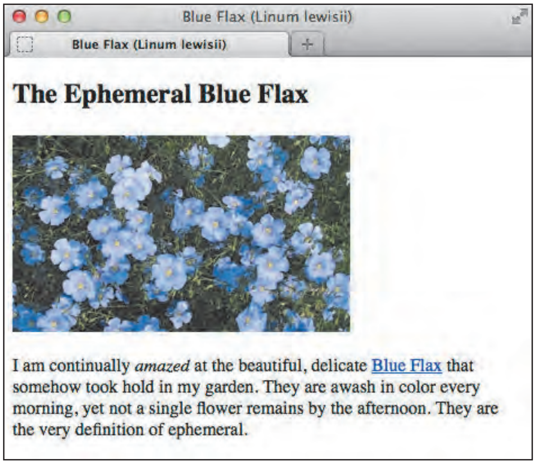


图 1.2.4　页面典型的默认呈现效果。这是页面在 Firefox 中显示的效果，在其他浏览器中的效果也是相似的。1.9 节解释了不同文本看起来不一样的原因

前面提到过（参见图 1.2.2），使用浏览器查看网页时，不会显示包围文本内容的标记。不过这些标记是非常有用的，我们就是使用它们来描述内容的。例如，`<p>` 标记用于表示段落的开始，具体的解释参见 1.8 节。

> **对 HTML 进行缩进**
>
> 代码的缩进（参见图 1.2.3）与内容在浏览器中的显示效果（参见图 1.2.4）没有任何关系（第 4 章会讲到，pre 元素是一个例外）。不过，对嵌套在父元素中的代码进行缩进是一种惯例，这样我们在编写和阅读代码时就会更容易看出元素之间的层级关系。

HTML 提供了很多这样的元素。图 1.2.3 所示的例子演示了六种最为常见的元素：a、article、em、h1、img 和 p。每个元素都有各自的含义，例如，h1 是标题，a 是链接，img 是图像。

我们暂时不会对示例代码做过多的探讨。在此之前，要进一步介绍 HTML 的基础知识，如通常意义上的元素、属性、文件名、URL 等。介绍完这些，再回过头来解释为什么要以这样的方式标记这些内容。我们还会介绍浏览器在默认情况下如何显示这些网页。

1.3　标签：元素、属性、值及其他

见识了一些 HTML 之后，我们来仔细看看标签的组成：元素（element）、属性（attribute）和值（value）。我们还将讨论父元素和子元素的概念。你在上文的基本 HTML 页面中已经见到了上面这些概念的例子，不过你可能并没有意识到。

1. 元素

我们在便利贴的例子中讲到，元素就像描述网页不同部分的小标签一样：这是一个标题，那是一个段落，而那一组链接是一个导航。我们在前面已经讨论过一些元素，有的元素有一个或多个属性，属性用来进一步描述元素。

大多数元素既包含文本，也包含其他元素（就像基本页面中的 p 元素）。如前面提到的，这些元素由开始标签、内容和结束标签组成。开始标签是放在一对尖括号中的元素的名称及可能包含的属性，结束标签是放在一对尖括号中的斜杠加元素的名称（参见图 1.3.1）。

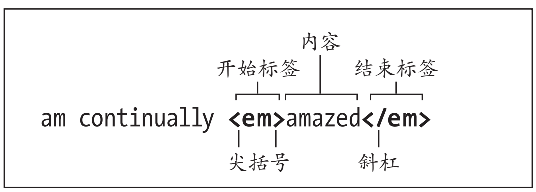


图 1.3.1　这是一个典型的 HTML 元素。开始标签和结束标签包着描述元素的文字。习惯上，标签采用小写字母

还有一些元素是空元素（empty element 或 void element），既不包含文本也不包含其他元素。它们看起来像是开始标签和结束标签的结合，由左尖括号开头，然后是元素的名称和可能包含的属性，然后是一个可选的空格和一个可选的斜杠，最后是必有的右尖括号（参见图 1.3.2）。

```
<img src="blueflax.jpg" width="300" height="175" alt="Blue Flax" />
                                                               可选的空格和斜杠
```

图 1.3.2　空元素（如这里显示的 img 元素）并不包含任何文本内容（alt 属性中的文字是元素的一部分，并非显示在网页中的内容）。空元素只有一个标签，同时作为元素的开始标签和结束标签使用。结尾处的空格和斜杠在 HTML5 中是可选的。不过，元素最后面的 > 是必需的

在 HTML5 中，空元素结尾处的空格和斜杠是可选的。XHTML 要求空元素结尾处必须有斜杠。以前用 XHTML 的人恐怕仍然倾向于在 HTML5 中继续使用斜杠，而其他人可能已经不用了。不管选择哪种方式，建议始终保持一致。

按照惯例，元素的名称都用小写字母。不过，HTML5 对此未做要求，也可以使用大写字母。只是现在已经很少有人用大写字母编写代码了，因此，除非无法抗拒，否则不推荐使用大写字母，这是一种过时的做法。

2. 属性和值

属性包含了元素的额外信息（参见图 1.3.3 和图 1.3.4）。在 HTML5 中，属性值两边的引号是可选的，但习惯上大家还是会写上，因此建议始终这样做。跟元素的名称一样，尽量使用小写字母编写属性的名称。

```
       for 是 label 的一个属性
<label for="email">Email Address</label>
            for 属性的值
```

图 1.3.3　这是一个 label 元素（关联文本标签与表单字段），它有一个简单的属性 – 值对。属性总是位于元素的开始标签内，属性的值通常放在一对括号中

```
  href 是 a 的一个属性                                    rel 也是 a 的一个属性
                        href 的值                             rel 的值
<a href="http://en.wikipedia.org/wiki/Linum_lewisii" rel="external"
→title="Learn more about Blue Flax">Blue Flax</a>
                    title 的值
  title 也是 a 的属性
```

图 1.3.4　有的元素（比如图 1.3.2 中的 a 和 img）可以有多个属性，每个属性都有各自的值。属性的顺序并不重要。不同的属性 – 值对之间都用空格隔开

本书会对大多数属性可接受的值进行详细说明，不过不妨先看看都有哪些类型的值。

有的属性可以接受任何值，有的则有限制。最常见的还是那些仅接受预定义值（也称为枚举值）的属性。也就是说，必须从一个标准列表中选一个值（参见图 1.3.5）。一定要用小写字母编写枚举值。

```
<link rel="stylesheet" media="screen" href="style.css" />
```

预定义值（screen）　非预定义值（style.css）

图 1.3.5　有的属性只接受特定的值。例如，只能将 `link` 元素里的 `media` 属性设为 `all`、`screen`、`print` 等值中的一个（参见第 8 章），不能像 `href` 属性和 `title` 属性那样可以输入任意值

有许多属性的值需要设置为数字，特别是那些描述大小和长度的属性。数字值无需包含单位，只需输入数字本身。图像（参见图 1.3.2）和视频的宽度和高度是有单位的，默认为像素。

有的属性（如 `href` 和 `src`）用于引用其他文件，它们只能包含 URL（统一资源定位符，是万维网上文件的唯一地址）形式的值。本章后面会详细介绍 URL。

最后，还有一种特殊的属性称为布尔属性（Boolean attribute）。这种属性的值是可选的，因为只要这种属性出现就表示其值为真（图 1.3.6）。如果一定要包含一个值，就写上属性名本身（这样做的效果是一样的）。布尔属性也是预先定义好的，无法自创（你应该没那么叛逆吧）。

```
<input type="email" name="emailaddr" required />
```

布尔属性（required）

图 1.3.6　这段代码提供了一个让用户输入电子邮件地址的输入框（参见第 18 章）。布尔属性 `required` 代表用户必须填写该输入框。布尔属性不需要属性值，如果一定要加上属性值，则写作 `required="required"`

3. 父元素和子元素

如果一个元素包含另一个元素，它就是被包含元素的父元素，被包含元素称为子元素。子元素中包含的任何元素都是外层的父元素的后代（参见图 1.3.7）。这种类似家谱的结构是 HTML 代码的关键特性，它有助于在元素上添加样式（从第 7 章开始讨论）和应用 JavaScript 行为（不在本书介绍范围）。

值得注意的是，当元素中包含其他元素时，每个元素都必须嵌套正确，也就是子元素必须完全地包含在父元素中。使用结束标签时，前面必须有跟它成对的开始标签。换句话说，先开始元素 1，再开始元素 2，就要先结束元素 2，再结束元素 1（参见图 1.3.8）。

```
<article>
   <h1>The Ephemeral Blue Flax</h1>
   <img src="blueflax.jpg"... />
   <p>... continually <em>amazed</em> ... delicate <a ...>Blue Flax</a> ...</p>
</article>
```

图 1.3.7 在这段 HTML 代码中，article 元素是 h1、img 和 p 元素的父元素。反过来，h1、img 和 p 元素是 article 元素的子元素（也是后代）。p 元素是 em 和 a 元素的父元素。em 和 a 元素是 p 元素的子元素，也是 article 元素的后代（但不是子元素）。反过来，article 元素是它们的祖先

```
      正确（代表标签对的线没有重叠）
<p>... continually <em>amazed</em> ...</p>
<p>... continually <em>amazed ...</p></em>
      错误（代表标签对的线交叉了）
```

图 1.3.8 元素必须正确地嵌套。如果先开始 p，再开始 em，就必须先结束 em，再结束 p

1.4 网页的文本内容

元素中包含的文本可能是网页上最基本的成分。如果你用过文字处理软件，那么你一定输入过文本。不过，HTML 页面中的文本有一些重要的差异。

首先，浏览器呈现 HTML 时，会把文本中的多个空格或制表符压缩成单个空格，把回车符和换行符转换成单个空格，或者将它们一起忽略（参见图 1.4.1 和图 1.4.2）。

其次，HTML 过去只能使用 ASCII 字符。ASCII 只包括英语字母、数字和少数几个常用的符号。开发人员必须用特殊的字符引用来创建重音字符（在很多西欧语言中很常见）和许多日常符号，如 é（表示 é）、©（表示 ©）等。完整列表见 www.elizabethcastro.com/html/extras/entities.html。

```
...
<body>
<p>I am continually <em>amazed</em> at
→ the beautiful,    delicate Blue Flax
→ that somehow took hold in my garden.

They are awash in       color every
→ morning, yet not a single flower
→ remains by the afternoon.


They are the very definition of
→ ephemeral.</p>
<p><small>&copy; Blue Flax Society.</small>
→ </p>
</body>
</html>
```

图 1.4.1 页面的文本内容（粗体部分）几乎就是标签以外的所有东西。在这个例子中，注意每个句子都至少包含一个回车符，有的词之间隔了好几个空格（在此只是为了说明 HTML 如何处理回车符和空格）。另外，第二段还包含一个表示版权符号的特殊字符引用（©）

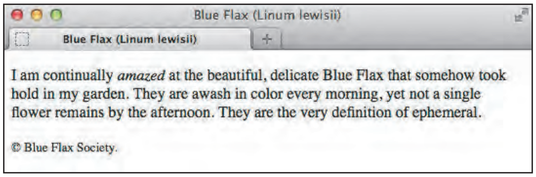


图 1.4.2　注意，使用浏览器查看这份文档时，多余的回车和空格都被忽略了，字符引用也替换成了对应的符号（©）

Unicode 极大缓解了特殊字符问题。用 UTF-8 对页面进行编码（参见图 1.4.3），并用同样的编码保存 HTML 文件（参见 2.3 节）已成为一种标准做法。推荐你也这样做。在图 1.4.3 中将 charset 值指定为 UTF-8 和 utf-8 的结果是一样的。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Blue Flax (Linum lewisii)</title>
</head>
<body>
...
</body>
</html>
```

图 1.4.3　直接在 head 开始标签后面指明文档的字符编码。charset 属性用来设置编码类型（通常为 UTF-8）

有时还是会使用字符引用，如版权符号（因为 © 既好记又好输入），参见图 1.4.1。

1.5　链接、图像和其他非文本内容

显然，万维网充满生机的部分原因是页面之间的链接，以及图像、视频、音乐等。外部文件（如图像）实际上并没有放在 HTML 文件中，而是单独保存的，页面只是简单地引用了这些文件（参见图 1.5.1）。

```
...
<body>
<article>
    <h1>The Ephemeral Blue Flax</h1>

    <img src="blueflax.jpg" width="300"
    → height="175" alt="Blue Flax" />

    <p>I am continually <em>amazed</em> at
    → the beautiful, delicate <a href=
    → "http://en.wikipedia.org/wiki/Linum_
    → lewisii" rel="external" title=
    → "Learn more about the Blue Flax">
    → Blue Flax</a> that somehow took
    → hold in my garden. They are awash
    → in color every morning, yet not a
    → single flower remains by the
    → afternoon. They are the very
    → definition of ephemeral.</p>
</article>
</body>
</html>
```

图 1.5.1　在我们的基本 HTML 文档中，有一个对图像文件 blueflax.jpg 的引用（位于 img 标签的 src 属性），浏览器在加载页面其他部分的同时，会请求、加载和显示这个图像。该页还包括一个指向关于 Blue Flax 的维基百科页面的链接（位于 a 标签的 href 属性）

浏览器可以轻而易举地处理链接和图像（参见图 1.5.2）。然而，它们无法处理其他任何文件类型。例如，对有的浏览器来说，要查看 PDF 就需要在系统中预先装好 Adobe Reader，要查看电子表格就需要预先装好 OpenOffice 这样的软件。

过去，HTML 没有内置的方法播放视频和音频文件。结果，各个厂商都开发出了相应的软件，用户可以下载并安装这些软件，从而弥补缺失的功能。这样的软件称为**插件**（plugin）。

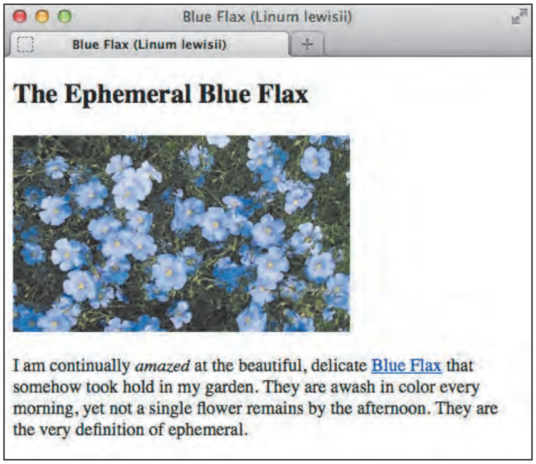


图 1.5.2 从网页引用图像和其他非文本内容，浏览器会将这些内容与文本一起显示。你在前面已经看到过了，默认情况下，链接文本的颜色与其他文本的颜色是不一样的，而且还带有下划线

在这些插件中，使用最为广泛的当数Flash。多年以来，Flash驱动了无数万维网上的视频。不过，这个插件也有一些问题，其中最为突出的就是它会耗费较多的计算资源。

幸好，HTML5已经为缓解这一问题迈出了一大步。HTML5提供了`audio`和`video`元素，这样，无需使用插件就可以播放音频和视频。不过，现代浏览器也提供了内置的媒体播放器，你仍然可以使用Flash播放器作为旧浏览器的备用工具。HTML5的音频和视频还不完美，但至少开始向着无插件的目标前进了，而且还在继续前进。

本书将在第5章讨论图像，在第6章讨论链接，在第17章讨论HTML5的音频和视频。

1.6 文件名和文件夹名

同其他文本文件一样，网页也有文件名。对网页文件命名时要记住几点，这些要点有助于对文件进行组织，使访问者更容易找到并访问你的页面，确保他们的浏览器能正确地处理页面，以及增强搜索引擎优化（SEO），参见图1.6.1和图1.6.2。（注意，本书会替换使用“文件夹”和“目录”，它们表示的含义相同。）



图 1.6.1 记住，文件名全部使用小写字母，用短横线分隔单词，用.html作为扩展名。混合使用大小写字母会增加访问者输入正确地址以及找到页面的难度

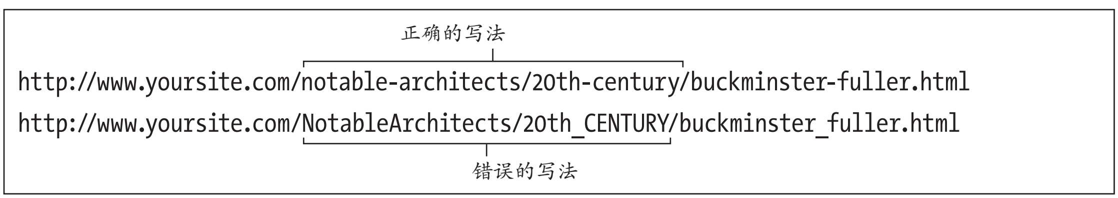


图 1.6.2 文件夹的名称也应全部用小写字母。关键是保持一致。如果使用小写字母，访问者和创建者就不必在大写字母和小写字母之间转换浪费时间了

1. 文件名采用小写字母

网页的文件名决定了访问者在访问你的页面时需要输入的文本，因此，文件名和文件夹名只用小写字母可以最大程度地避免访问者的输入错误。这样做对创建页面之间的链接也有很大的帮助。如果所有的文件名都用小写字母，你就少了一件需要操心的事情。

2. 使用正确的扩展名

浏览器主要通过查看文件的扩展名得知需要读取的文本文档是一个网页。尽管也可以使用 .htm 作为网页的扩展名，但通常使用的是 .html。如果页面使用其他的扩展名（如 .txt），浏览器会将其当做文本处理，相当于将你的代码直接呈现给访问者。

3. 用短横线分隔单词

不要在文件名和文件夹名中使用空格分隔单词。应该使用短横线，例如 company-history.html 和 my-favorite-movies.html。有的网站使用下划线（_），我们不推荐这种做法，因为短横线是搜索引擎更容易接受的方式。

提示　SEO 可以让网页在搜索引擎的搜索结果中排名靠前。

提示　Mac OS 和 Windows 有时不会显示文档的扩展名。如需查看扩展名，请更改文件夹选项。

1.7　URL

URL（Uniform Resource Locator，统一资源定位符）是地址的别名。它包含关于文件存储位置和浏览器应如何处理它的信息。互联网上的每个文件都有唯一的 URL。

URL 的第一个部分称为**模式**（scheme）。模式告诉浏览器如何处理需要打开的文件。最常见的模式是 HTTP（Hypertext Transfer Protocol，超文本传输协议）。根据你上网的经验，你应该可以猜到，HTTP 是用于访问网页的（参见图 1.7.1）。HTTPS 是从 HTTP 衍生来的，用于电子商务网站等安全网页。它的格式同 HTTP 一样，只是用 https 替代了 http。

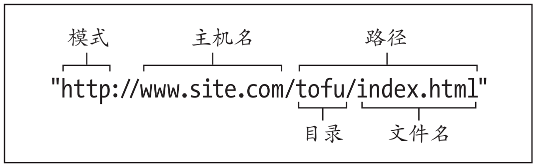


图 1.7.1　基本的 URL 包含模式、主机名称和路径。路径包含一个或多个目录（文件夹），最后是一个文件名

URL 的第二个部分是文件所在的主机的名称，紧接着是路径，路径包含到达这个文件的文件夹以及文件自身的名称（这些都是可选的）。如果 A 中的路径是 tofu/soft/index.html，就表示 index.html 位于 soft 目录，而 soft 目录则位于 tofu 目录，就像你在自己的电脑上组织文件和文件夹那样。

有时，URL 路径不以文件名结尾，而以一个目录结尾（可以包含一个结尾的斜杠，也可以不包含），如图 1.7.2 所示。在这种情况下，URL 指的是路径中最后一个目录中的默认文件，通常为 index.html。（通常情况下，所有 Web 服务器的配置都将 index.html 作为默认文件名，因此你不必修改任何服务器配置。）



图 1.7.2　以一个斜杠而非文件名结尾的 URL 指向最后一个目录（在这个例子中是 tofu 目录）中的默认文件。最常见的默认文件名是 index.html。因此，这个 URL 与上一个例子中的 URL 指向的是同一个页面

其他常见的模式有用于下载文件的 ftp（File Transfer Protocol，文件传输协议），如图 1.7.3 所示；用于发送电子邮件的 mailto，如图 1.7.4 所示（参见第 6 章）。

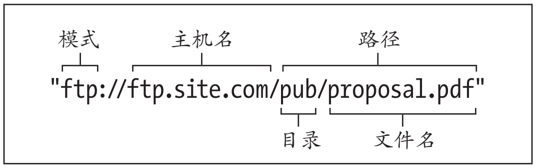


图 1.7.3　当用户点击这个 URL 时，浏览器会开始按 FTP 模式传输文件 proposal.pdf（有时需要先输入用户名和密码）

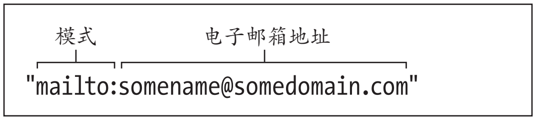


图 1.7.4　用于电子邮件地址的 URL 包括 mailto 模式，紧接着是一个冒号（没有斜杠），然后是电子邮箱地址本身

模式后面通常紧跟一个冒号和两个斜杠。mailto 是个例外，它后面只有一个冒号。

在上述模式中，最常用到的是 http（也包括 https），其次是 mailto，再次是 ftp。其他模式则只有在特殊情况下才会用到，你很少会碰到它们。

1. 绝对 URL

URL 可以是绝对的，也可以是相对的。绝对 URL（absolute URL）包含了指向目录或文件的完整信息，包括模式、主机名和路径（图 1.7.1 和图 1.7.2）。绝对 URL 就像是完整的通信地址，包括国家、州、城市、邮政编码、街道和姓名。无论邮件来自哪里，邮局都能找到收件人。就 URL 来说，这意味着绝对 URL 本身与被引用文件的实际位置无关，无论是在哪个主机上的网页中，某一文件的绝对 URL 都是完全一样的。

引用别人 Web 服务器上的文件时，应该总是使用绝对 URL。当你通过电子邮件跟你的朋友分享新闻或 YouTube 视频的 URL 时，如果只给他们 URL 的一部分，他们就无法看到相应的内容。这也就是为什么我们的基本页面（参见图 1.7.5）中指向维基百科页面的 href 值是完整的 URL 而不是简单的 Linum_lewisii。

```
<p>I am continually <em>amazed</em> at
→ the beautiful, delicate <a href="http://
→ en.wikipedia.org/wiki/Linum_lewisii"
→ rel="external" title="Learn more about the
→ Blue Flax">Blue Flax</a> that somehow took
→ hold in my garden . . .</p>
```

图 1.7.5　由于我们的基本页面并不在 en.wikipedia.org，因此在链接到关于 Blue Flax（拉丁文中为 Linum lewisii）的页面时，需要使用绝对 URL。第 6 章对链接作了详尽的介绍，包括 rel 属性（如果链接是指向其他网站的，推荐使用这一属性）

对于 FTP 站点以及几乎所有不使用 HTTP 协议的 URL，都应该使用绝对 URL。

2. 相对 URL

当我告诉你我邻居家的位置时，我一般不会说完整地址，而是说：“她家在右边第三个门。”这就是相对地址，它指出的位置是以信息提供者的位置为参照的。如果在别的城市按照同样的信息找我的邻居，你永远也找不到。（实际情况是，她出门很长一段时间了，因此恐怕你还是找不到她。）

同样，相对 URL 以包含 URL 本身的文件的位置为参照点，描述目标文件的位置。因此，相对 URL 可以表达像“指向本页面同一目录下的 xyz 页面”这样的意思。

◎ 引用同一目录下的文件

如果目标文件与当前页面（也就是包含 URL 的页面）在同一个目录中，那么这个文

件的相对 URL 就只有文件名和扩展名（参见图 1.7.6）。例如，链接的 HTML 可能是 Take me to history.html!。

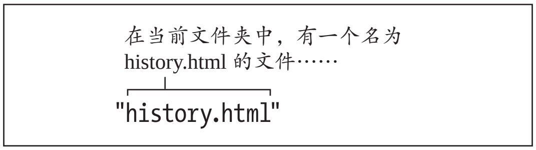


图 1.7.6　相对 URL 指向同一目录下的文件（参见图 1.7.10）。URL 中只需要文件名和扩展名，不必在前面加上 http://www.site.com/about/（这两个文件所在的主机名和目录）

◎ 引用子目录下的文件

如果目标文件在当前目录的子目录中，那么这个文件的相对 URL 就是子目录名，接着是一个斜杠，然后是文件名和扩展名（参见图 1.7.7）。例如，Data supports my hypothesis。

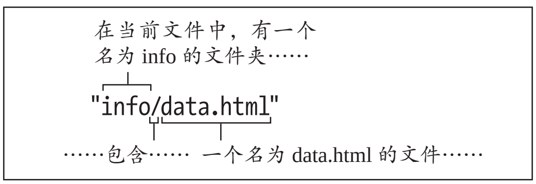


图 1.7.7　要引用当前文件夹的子文件夹中的文件，在这个例子中是 data.html（参见图 1.7.10），应在文件名之前加上子文件夹名称和一个斜杠

◎ 引用上层目录的文件

如果要引用文件层次结构中更上层目录中的文件，那么应该使用两个句点和一个斜杠（参见图 1.7.8），例如，our products。每个 ../ 都表示“到当前文件的上一层”，因此，../../ 会向上走两级，../../../ 会向上走三级。

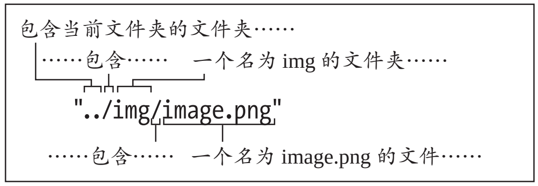


图 1.7.8　从图 1.7.10 中可以看到，这个文件位于与网站根目录下的当前文件夹（about）同属一层的文件夹（img）中。在这种情况下，使用两个句点和一个斜杠上升一级，然后指出子目录（img），再跟一个斜杠，最后是文件名（实践中，可以为 image.png 选择一个更具描述性的文件名，这里的名称仅作为示例）

◎ 根相对 URL

不过，如果文件位于 Web 服务器上，应该避免使用像 ../../img/family/vacation.jpg 这样显得较为笨拙的文件路径，不应先跳到网站的根目录再顺着向下找到目标文件（参见图 1.7.9）。可以在最开始使用一个斜杠，这样本例中的**根相对** URL 就是 /img/family/vacation.jpg（假定 img 文件夹位于网站的根文件夹，这也是惯常的用法）。需要强调的是，这种做法只能用于 Web 服务器，例如在网站托管服务供应商的 Web 服务器上，或者在本地计算机运行的 Web 服务器上（参见提示）。

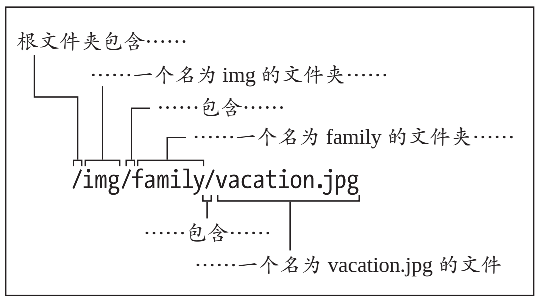


图 1.7.9　相同的根相对 URL 可用于所有的页面，不管页面在网站目录结构中的具体位置。例如，在这个例子中，主页和六层目录中某个页面对 vacation.jpg 的定位是同样简单的（这并不代表我建议使用这么深的层级结构）

如果不是在服务器本地开发网站，通常应使用相对 URL（当然，除非是指向其他服务器上的文件）。这样确保了将页面从本地系统传到服务器变得容易。只要每个文件的相对位置保持不变，就不必修改任何路径，链接依然有效。

提 示　Apache 是在计算机上运行开发服务器最常见的选择。可以在网上搜索“set up local dev environment”（创建本地开发环境）。

提 示　第 21 章会讨论如何找到一个 Web 主机。

3. 绝对 URL 和相对 URL 的比较

为了巩固上述知识，表 1.7.1 和图 1.7.10 共同演示了绝对 URL 和相对 URL 的区别。图 1.7.10 显示了两个网站的文件和目录结构。表 1.7.1 描述了从 you-are-here.html 访问各个不同文件的方式，这些文件有的位于同一个网站（www.site.com），有的位于另一个网站（www.remote.com）。尽管表格的前三行也可以使用绝对 URL，但访问同一网站的文件时最好使用相对 URL。

表 1.7.1　绝对 URL 与相对 URL

文 件 名	绝对 URL（可以在任何地方使用）	相对 URL（只能在 you-are-here.html 中使用）
history.html	http://www.site.com/about/history.html	history.html
data.html	http://www.site.com/about/info/data.html	info/data.html
image.png	http://www.site.com/img/image.png	../img/image.png
news.html	http://www.remote.com/press/news.html	（无，使用绝对 URL）
join.html	http://www.remote.com/sign-up/join.html	（无，使用绝对 URL）

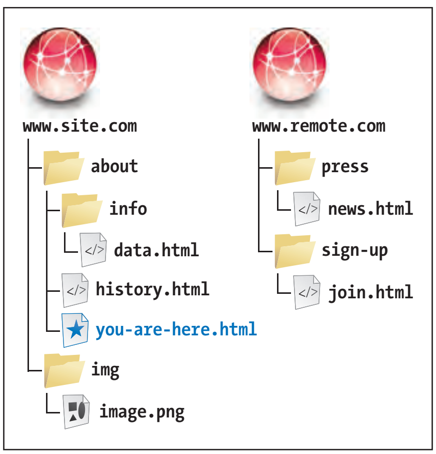


图 1.7.10　包含 URL 的文档（在这个例子中是 you-are-here.html）是相对 URL 的参照点。换句话说，相对 URL 是相对于这个文件在服务器上的位置。绝对 URL 与其所在的位置无关，因为它们总是包含资源的完整 URL

1.8　HTML：有含义的标记

本章一开始便将在网页中标记内容比作为橱柜物品打标签，因此你已经了解到了 HTML 的功能和相关的基础知识。

这里强调的是，HTML 描述的是网页内容的含义，即**语义**（semantics）。在 Web 社区中，**语义化 HTML**（semantic HTML）指的是那些使用最恰当的 HTML 元素进行标记的内容，在标记的过程中并不关心内容显示。我想你应该同意这种做法，这比胡乱选择元素要有意义得多。

更好的是，这样做也非常容易，就像下面对我们的基本网页所做的解释那样。在此之后，“为什么语义很重要”进一步强调了编写语义化 HTML 是构建有效网站的基石。

1. 基本 HTML 页面的语义

本章前面提到，我们会回过头来讲解基本页面，接下来就来做这个工作。给你一分钟时间整理一下思路。

好了，接下来要深入地了解标记内容背后的思维过程。在此需要了解一些最为常用的 HTML 元素（参见图 1.8.1），所有这些元素都会在后续章节进行详细阐述。正如你看到的，创建拥有良好语义的 HTML 并不难。一旦熟悉这些元素，这一过程就会变得很简单了。

```
...
<body>
<article>
    <h1>The Ephemeral Blue Flax</h1>

    <img src="blueflax.jpg" width="300"
    → height="175" alt="Blue Flax" />

    <p>I am continually <em>amazed</em> at
    → the beautiful, delicate <a href=
    → "http://en.wikipedia.org/wiki/Linum_
    → lewisii" rel="external" title="Learn
    → more about Blue Flax">Blue Flax</a>
    → that somehow took hold in my garden.
    → They are awash in color every morning,
    → yet not a single flower remains
    → by the afternoon. They are the very
    → definition of ephemeral.</p>
</article>
</body>
</html>
```

图 1.8.1 基本页面的 body 元素包含了 article、h1、img、p、em 和 a 元素。所有的内容都包含在 article 元素中

所有的内容都包含在 article 元素中，如图 1.8.1 所示。简言之，article 定义了一段独立的内容，其他地方可以重用这段内容。在我们的基本网页中使用 article 元素是个不错的选择，但并非每个网页都要这样编写。第 3 章会详细阐释 article 元素。

接下来是标题（参见图 1.8.2）。HTML 提供了六个标题级别，即 h1 ~ h6。其中，h1 是最重要的一级，h2 是 h1 的子标题，h3 是 h2 的子标题，以此类推，就像你使用文字处理软件编辑具有多级标题的文档一样。

```
<h1>The Ephemeral Blue Flax</h1>
```

图 1.8.2 标题是描述页面概貌的重要元素。标题确保了页面对屏幕阅读器用户来说更具可访问性，而搜索引擎用它们确定页面的重点

每个 HTML 都应该有一个 h1（或者多个 h1，这取决于页面的内容）。我们的页面只有一个标题，显然应该使用 h1。第 3 章会详细介绍 h1 ~ h6。

接下来，是一个图像（参见图 1.8.3）。img 元素是呈现图像的主要方式，对此，无需讨论用哪个元素合适。如果图像没有加载成功，或者页面是通过仅显示文本的浏览器查看的（这种情况已经很少见了），就会显示 alt 属性中的文字。屏幕阅读器可以对 alt 文本进行朗读（参见后面的“无障碍访问”）。第 5 章会详细介绍图像相关的内容。

```
<img src="blueflax.jpg" width="300"
→ height="175" alt="Blue Flax" />
```

图 1.8.3 用 img 在页面中插入图像很容易

段落使用 p 元素进行标记，如图 1.8.4 所示。正如印刷材料中的段落所显示的，一个段落可以包含多个句子。如果页面需要再加一个段落，只需要在第一个 p 元素之后再加一个 p 元素就可以了。

段落中嵌套了两个元素（em 和 a），分别定义其短语内容的含义（图 1.8.4）。这是 HTML5 提供的大量用于提升段落文本语义的**短语内容**（phrasing content）元素的两个例子。我们会在第 4 章中讨论短语内容元素和 p 元素。

```
<p>I am continually <em>amazed</em> at
➝ the beautiful, delicate <a href="http://
➝ en.wikipedia.org/wiki/Linum_lewisii"
➝ rel="external" title="Learn more about
➝ Blue Flax">Blue Flax</a> that somehow
➝ took hold in my garden. They are awash in
➝ color every morning, yet not a single
➝ flower remains by the afternoon. They are
➝ the very definition of ephemeral.</p>
```

图 1.8.4 p 元素可以包含定义段落内短语语义的其他元素，em 和 a 是两个例子

em 元素表示“强调”，就像你在说话时会重读某些词语。在我们的示例页面中，它强调了对花的惊叹之情（图 1.8.4）。记住，HTML 描述的是内容的含义，因此 em 代表的是语义上的强调，而非视觉上的，不过通常会用斜体表示 em 文本（可以使用 CSS 改变这一样式）。

最后，基本页面通过 a 元素定义了一个链接。a 表示“锚”（anchor）。链接可谓是所有 HTML 元素中最强大的元素，因为有它才得以形成万维网。万维网的定义就是，将一个页面与另一个页面或资源连接起来，或者将页面的一部分与另一部分连接起来。在这个例子中，文字“Blue Flax”是一个指向维基百科某页面的链接（参见图 1.8.5）。

```
<a href="http://en.wikipedia.org/wiki/Linum_lewisii" rel="external" title="Learn more about Blue Flax">
➝ Blue Flax</a>
```

图 1.8.5 a 元素定义了一个指向维基百科中关于 Blue Flax 的链接。可选的 rel 属性指出链接指向的是另一个网站，这也增加了语义（没有这个属性链接也会正常工作）。可选的 title 属性提供了有关所指页面的信息，它也增强了 a 元素的语义。当用户用鼠标滑过该链接，就会显示 title 属性的内容

相当简单，对吧？当你对 HTML 元素了解更多之后，为内容选择正确的元素通常是相当简单的工作。偶尔，你会遇到一些内容，有一种以上的合理标记方式，这也没问题。有时候不能直接辨别对与错，当然大多数时候可以。

无论如何，浏览器都会将内容显示出来。浏览器还没有聪明到能够判断一段内容更适合使用 p 元素还是其他元素。

最后需要指出，HTML5 并没有试图为每一种能想到的内容类型提供对应的元素，因为这样会使这门语言变得笨重。相反，HTML5 采取了一种务实的态度，其所定义的元素覆盖了绝大多数情况。

HTML 之美，一部分在于人们能很容易掌握它的基础，构建一些页面，并在此基础上不断取得进步。因此，尽管 HTML 元素大约有 100 种，但不要被这个数字吓到，你经常用到的只有少量核心元素，而其余的则较少使用。

你已经了解了一些常见元素，因此你已经有了一个良好的开始。

2. 为什么语义很重要

这里列出了使用语义化 HTML 最重要的几个原因。我们没有给出全部原因，我们已经在前面提到一些内容了。

- ❑ 提升可访问性和互操作性（内容对于借助辅助技术的残障访问者是可访问的，同时对于台式机、手机、平板电脑及其他设备上的浏览器都是可访问的）。
- ❑ 提升搜索引擎优化（SEO）的效果。
- ❑ 使维护代码和添加样式变得容易。
- ❑ （通常）使代码更少，页面加载更快。

◎ 无障碍访问

你可能对无障碍访问并不熟悉。它指

的是让内容对所有用户可用，不论其能力如何（参见 www.w3.org/standards/webdesign/accessibility）。万维网的发明者 Tim Berners-Lee 曾说过一句著名的话："万维网的力量在于其普适性。让包括残障人士在内的每个人都能访问万维网，是极为重要的一点。"

任何带有浏览器的设备都可以显示 HTML，因为它只是文本。然而，用户获取内容的方式可能不同。例如，视力正常的人可以直接查看内容，而视力受损的用户则需要放大页面，调大字号，或者使用屏幕阅读器（可以将内容朗读出来的软件，是辅助技术的一个例子）。

有时，屏幕阅读器会将内容周围的 HTML 元素的类型读出来，让用户了解上下文。例如，对于列表，在读出列表各个条目之前，屏幕阅读器会首先告诉用户这里有一个列表。类似地，对于链接，屏幕阅读器会告诉用户这里有一个链接，方便其决定是否点击这个链接。

屏幕阅读器用户能够以多种方式浏览网页，例如通过键盘按键从一个标题跳到下一个标题。这样，他们可以先了解一个页面的关键主题有哪些，再去听他们感兴趣的内容，而不必把整个页面从头到尾听下来。

由此可见，对残障人士来说，好的语义产生了多么大的差别。

◎ 搜索引擎优化（SEO）

SEO 的效果也会得到改善，也就是说网页在搜索引擎中的排名会靠前，因为搜索引擎对用特殊方式标记的内容会赋予更高的权重。例如，标题告诉搜索引擎爬虫页面的主要主题，帮助浏览器对页面目录进行索引（index）。

◎ 更容易维护代码和添加样式

随着不断深入阅读本书，你会了解为什么好的语义能使代码更有效、更易于维护和添加样式。你将了解到，借助 CSS，我们可以轻松统一特定元素的样式，例如，让所有的段落显示为深灰色，并使用 Georgia 字体。如果你将某些段落标记为 p 元素，某些标记为其他元素，那么就要在 CSS 中同时为这两种元素设定样式，这就添加了不必要的复杂性。

这样做也会让维护 HTML 变得困难。好的语义可以让网页变得统一和"干净"。这样做还会使文件尺寸变小，从而浏览器加载网页的速度变得更快。

1.9　浏览器对网页的默认显示效果

既然控制页面样式的是 CSS 而非 HTML（参见图 1.9.1），为什么在浏览器中查看我们的基本页面时，某些文字的字号会比其他文字的字号大呢，为什么有的文字是粗体或斜体呢（参见图 1.9.2）？

原因就是每个 Web 浏览器都有一个内置 CSS 文件（一张**样式表**），它决定了每个 HTML 元素的默认样式。你自己创建的 CSS 可以覆盖这些样式。对于不同的浏览器，默认样式会稍有差异，但总体上相当一致。重要的是，HTML 所定义的内容的底层结构和含义是一致的。

块级元素、行内元素以及 HTML5

容易看出，有些 HTML 元素（如 `article`、`h1` 和 `p`）从新的一行开始显示，就像纸质书中的各个段落一样，而另外一些元素（如 `a` 和 `em`）则与其他内容显示在同一行，如图 1.9.2 所示。需要强调的是，这是浏览器的默认样式，而不是 HTML 元素自身的样式，也不是由代码中两个元素之间的空行引起的（参见图 1.9.1）。

```
...
<body>
<article>
    <h1>The Ephemeral Blue Flax</h1>

    <img src="blueflax.jpg" width="300"
    → height="175" alt="Blue Flax" />

    <p>I am continually <em>amazed</em> at
    → the beautiful, delicate <a href=
    → "http://en.wikipedia.org/wiki/Linum_
    → lewisii" rel="external" title="Learn
    → more about Blue Flax">Blue Flax</a>
    → that somehow took hold in my garden.
    → They are awash in color every morning,
    → yet not a single flower remains
    → by the afternoon. They are the very
    → definition of ephemeral.</p>

    <p><small>&copy; Blue Flax Society.
    → </small></p>
</article>
</body>
</html>
```

图 1.9.1　在基本页面的结尾又加了一段，这样在浏览器中查看网页时就能看到每个段落都占据单独的一行（图 1.9.2）。顺便说一下，small 元素表示的含义是法律声明等条文细则。默认情况下，它比其他的文字显示得小一些，但是显示小字号并不是使用这个元素的理由

这里需要多解释一下。在 HTML5 之前，大多数元素都可以划入**块级**（block-level，从新行开始显示）或**行内**（inline，与其他内容在同一行显示）两种类别。HTML5 废弃了这些术语，因为这些术语把元素与表现关联起来，而 HTML 并不负责表现。（通常，旧的行内元素在 HTML5 中都被归类为短语内容。）

尽管如此，浏览器不需要也不应该为这些元素改变默认显示规则。毕竟，你不想看到两个段落（p 元素）连到一起，或者强调的字句（em 元素包含的 amazed）将句子打断，单独成行。

因此，通常标题、段落和 article 这样的元素从新行开始显示，而短语内容（如 em、a 和 small）则与外围内容在同一行显示。



图 1.9.2　浏览器的默认样式表将标题（h1 ～ h6）与普通文本区别开来，对 em 文本加上斜体样式，对链接加上颜色和下划线。此外，有的元素从单独的一行开始（如 h1 和 p），而其他一些元素显示在外围内容的里面（如 a、em 和 small）。用你自己的样式表覆盖这些显示规则很简单

尽管 HTML5 不再使用块级、行内这些术语，但这样划分有助于理解它们的含义。由于在 HTML5 之前就成为 HTML 的常用词，因而这些术语在教程中很常见。本书也会偶尔使用这些术语，以说明元素在默认情况下是另起一行还是与其他内容共处一行。

从第 7 章开始将详细讲解 CSS，不过现在只需要知道：样式表与 HTML 一样是纯文本，因此可以用编辑 HTML 的文本编辑器创建样式表。下一章就会讲解如何使用编辑器操作 HTML 文件。

1.10　要点回顾

HTML 基础以及某些关键最佳实践为构建有效网站打下了基础。下面我们来回顾一下本章讲述的要点。

- 一个网页主要由三种成分构成：文本内容、对其他文件的引用和标记。
- HTML 标记由元素、属性和值构成。
- 通常全部使用小写字母编写 HTML（DOCTYPE 是一个例外），用引号包住属性值。
- 为文件和文件夹命名时全部采用小写字母，词与词之间用短横线分隔，而不要用空格或者下划线。
- 始终用下面的 DOCTYPE 声明开始 HTML 文档，让浏览器知道这是一个 HTML5 页面。

  ```
  <!DOCTYPE html>
  ```
- 页面内容都在 body 元素中。主要为浏览器和搜索引擎准备的指令位于 body 元素之前，在 head 元素中。
- 用语义化 HTML 标记内容，不关心它在浏览器中显示的样式。
- 语义化 HTML 提升了网站的可访问性，让网站更有效率，并使网站维护和添加样式变得更容易。
- CSS 控制 HTML 内容的表现。
- 每个浏览器自带的样式表规定 HTML 的默认表现样式。开发人员可以使用自己写的 CSS 覆盖这些规则。

第 2 章

处理网页文件

本章内容

- ❑ 规划网站
- ❑ 创建新的网页
- ❑ 保存网页
- ❑ 指定默认页面或主页
- ❑ 编辑网页
- ❑ 对文件进行组织
- ❑ 在浏览器中查看网页
- ❑ 借鉴他人灵感

开始编写 HTML 元素和属性代码之前，有必要了解如何创建使用这些代码的文件。本章将介绍如何创建、编辑和保存网页文件，还会涉及一些设计和组织方面的注意事项。

如果你已经知道如何创建文件，没有耐心读完本章，可以直接跳到讲解 HTML 代码的第 3 章。

2.1 规划网站

我们可以一上来就直接编写网页，但最好还是先对网站进行思考和规划。这样，你就不会迷失方向，而且也会减少将来的重组工作。比起简单地知道如何编写代码，了解如何创建有效的网站要更重要一些。下面介绍的内容并不全面，不过确已涵盖了不少需要考虑的方面。

规划网站的方法

- ❑ 确定为什么要创建这个网站，需要展示什么内容。
- ❑ 考虑网站的访问者。应该如何调整内容使之吸引这些访问者。
- ❑ 需要多少个页面，你希望网站是怎样的结构（参见图 2.1.1），你是希望访问者以某种特定的次序浏览网站，还是希望访问者可以自由地探索。

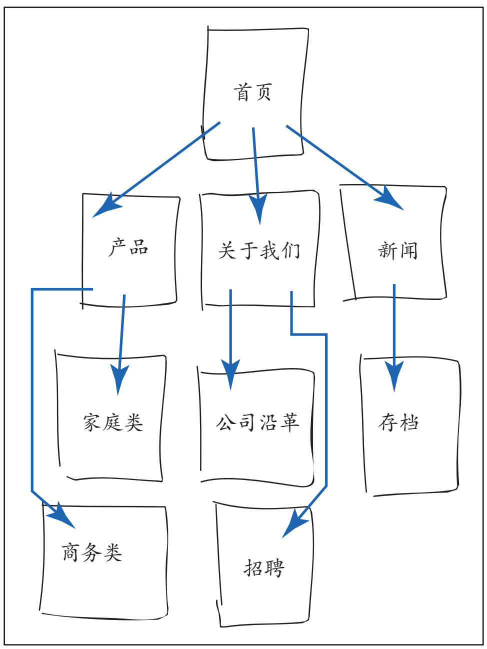


图 2.1.1 画出网站结构草图，考虑它可能包含的内容，这有助于创建者决定它需要何种结构

- ❑ 在纸上画出网站结构的草图，确定你在每个页面希望呈现的内容。与其他的工作不同，这项工作是能指导网站设计的。
- ❑ 为页面、图像和其他外部文件设计一个简单且一致的命名规则（参见 1.6 节）。

提 示　如果你对万维网还不太熟悉，先上网逛逛，了解可能的网站形式。可以从竞争对手的网站开始入手。

提 示　很多网站提供内容战略、用户体验（UX）、设计、开发等与建站相关的资源，其中最有名的两个是 A List Apart（www.alistapart.com）和 Smashing Magazine（www.smashingmagazine.com）。

提 示　Erin Kissane 的文章“A Checklist for Content Work”（www.alistapart.com/articles/a-checklist-for-content-work/）讲解了如何制作网站内容。这部分内容也被收录到了她关于内容战略的一本书中。

提 示　如果你还没有成为设计师，只是一位设计新手，正在寻找网站设计方面的指导，Jason Beaird 的《完美网页的视觉设计法则》（*The Principles of Beautiful Web Design*，SitePoint，2010）和 Mark Boulton 的《Web 设计实战》（*A practical Guide to Designing for the Web*，Five Simple Steps，2009）或许会引起你的兴趣。Boulton 的书最开始只有纸质书，现在又在网上发布了免费的版本（http://designingfortheweb.co.uk/book/）。这本书不仅讲解了设计理论，还介绍了 Web 设计者的工作方法。

提 示　虽然不是必需的，但是将网站的文件夹结构与网站在纸上的组织结构对应起来是很常见的做法（图 2.1.1），参见 2.6 节。

2.2　创建新的网页

创建网站并不需要特殊的工具。你可以使用任何文本编辑器，甚至是 Windows 自带的记事本 Notepad，如图 2.2.1 和图 2.2.3 所示，或者 OS X 上的免费软件 TextWrangler（www.barebones.com/products/textwrangler），如图 2.2.1 和图 2.2.2 所示。（Mac 也自带一款编辑器，名为 TextEdit，但它在 OS X 某些版本中存在漏洞，使之无法正确处理 HTML 文件。）

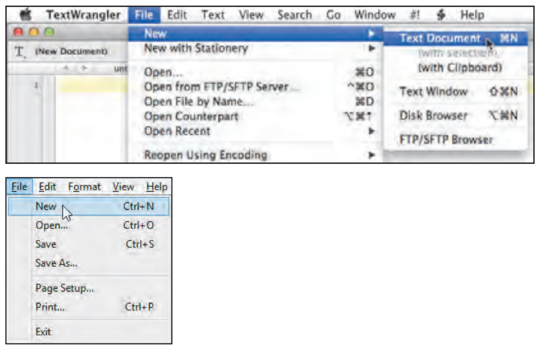


图 2.2.1　打开文本编辑器。在编辑器中出现的空白文档中输入 HTML，或者先选择 File → New。实际的菜单选项可能略有差异。如果使用 TextWrangler（Mac），就是 File → New → Text Document（文件→新建→文本文档），如上半部分所示。下半部分显示的是 Notepad（Windows）的选项

创建新网页的步骤

(1) 打开文本编辑器。

(2) 选择 File → New（文件→新建）创建一个新的空白文档，如图 2.2.1 所示。

(3) 按照本书其余部分的解释（从第 3 章开始）创建 HTML 内容。

(4) 一定要按照 2.3 节中的说明保存文件。

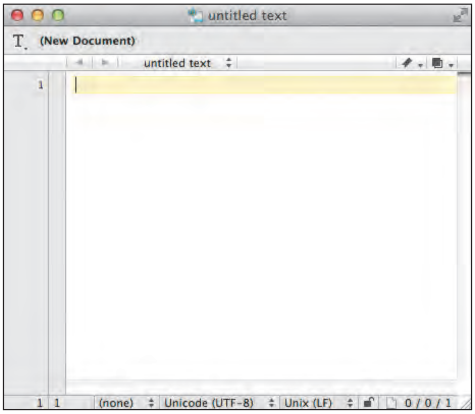


图 2.2.2　在 Mac 上，可以使用 TextWrangler 编写网页的 HTML 代码。下文提示部分列出了一些功能更强的编写代码的 Mac 编辑器

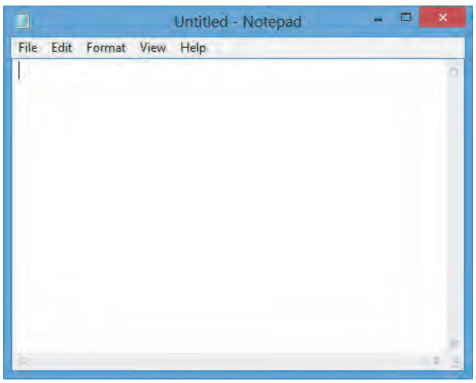


图 2.2.3　这是记事本，Windows 用户用以创建 HTML 页面的最基本的程序。也可以使用其他的编辑器（参见提示）

提 示　在 OS X 和 Windows 上都有很多专门编写 HTML（和 CSS）代码的文本编辑器。它们的代码提示和代码补全功能让你可以更准确、更快速地编写代码；它们会将代码突出显示，方便区分 HTML 元素和其中的文本内容；它们还有其他有用的功能。记事本没有这些功能。有不少免费的 HTML 编辑器，不过一些收费的编辑器也值得购买，通常还可以在购买之前试用。

提 示　OS X 的流行编辑器包括 BBEdit（www.barebones.com/products/bbedit/）、Coda（www.panic.com/coda/）、Espresso（http://macrabbit.com/espresso/）、Sublime Text（www.sublimetext.com）以及 TextMate（http://macromates.com）。通常可以认为 TextWrangler 是 BBEdit Lite 的简化版。Sublime Text 也有 Windows 版本，Windows 下的编辑器还有很多，如 Notepad++（http://notepad-plus-plus.org），等等。在网上搜索“HTML 编辑器”可以找到更多的编辑器。

提 示　使用上面列出的某种编辑器创建新网页的方法也是类似的。要编辑已经存在的页面，只需在文本编辑器中选择 File → Open（文件→打开）并打开该文件（参见 2.5 节）。

提 示　请不要使用 Microsoft Word 这样的文字处理软件编写 HTML 页面。它们会在文件里添加一些无用的或无效的代码。

2.3　保存网页

用文本编辑器创建的网页需要在多种平台和多种设备上的多种浏览器查看。为了让所有这些程序都能访问网页，网页需要保存为通用的“纯文本”格式，不包含文字处理软件可能应用的任何专用格式化信息。

为了让浏览器将 HTML 文档作为网页识别并知道解释其中包含的标记，网页文件应在文件名中使用 .html 或 .htm 作为扩展名。这样做也可以将网页文件与不是网页的普通文本文件区分开来。上述两种扩展名都可以，但通常还是使用 .html，因此推荐读者使用 .html 作为文件扩展名。

有了 .html 扩展名，网页的图标会显示为

系统默认浏览器的图标，而不是用来编写这个文件的编辑器的图标（如图 2.3.1 所示）。双击网页文件会在浏览器中打开它，而不是在文本编辑器中。这对于在浏览器中测试页面很方便，但却为编辑网页增加了一个额外的步骤（参见 2.5 节）。

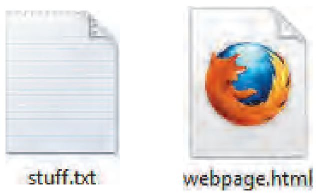


图 2.3.1　文本文件的扩展名是 .txt，在 Windows 上使用原生的文本文件图标进行标识（左图）。双击图标会在记事本中显示它的内容（如果你将文本文件跟其他的文本编辑器关联起来，就会启动那个软件）。对于网页文件（右图），不管是用哪个文本编辑器创建的，只要是以 .html 或 .htm 为扩展名，都会使用默认浏览器图标进行标识（在这里是 Firefox）。双击这个图标会在默认浏览器而不是文本编辑器中显示它

总之，保存文件时，需要将文件保存为纯文本格式，并使用 .html 或 .htm 作为扩展名。

保存网页的步骤

(1) 创建网页之后，在文本编辑器中选择 File → Save As（文件→另存为），如图 2.3.2 所示。

(2) 在随后弹出的对话框中，选择纯文本或文本文档（或别的叫法）作为文件格式。

(3) 为文档添加 .html 或 .htm 的扩展名（这一点非常重要）。

(4) 选择要保存网页的文件夹。

(5) 点击 Save（保存），如图 2.3.3 和图 2.3.4 所示。

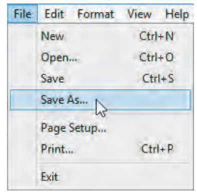


图 2.3.2　在文本编辑器中选择 File → Save As，这里显示的是记事本。TextWrangler 也有这个选项，不过位于一个更长的菜单中

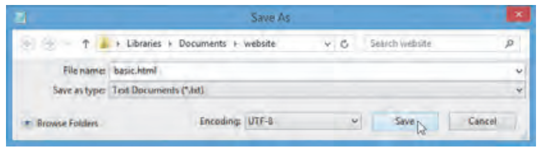


图 2.3.3　在记事本中，为文件名添加 .html 或 .htm 的扩展名，在 Save as type（保存类型）下拉菜单中选择文本文档，确保 Encoding（编码）选择的是 UTF-8（参见最后一项提示），点击 Save。其他文本编辑器中的选项可能并不相同（不过是相似的）

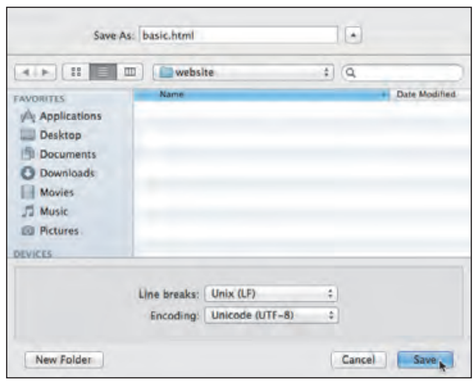


图 2.3.4　在 TextWrangler 中，为文件起名，选择保存位置。TextWrangler 默认以 UTF-8 进行编码（一般情况下，这就是应该选用的编码），不过你也可以从 Encoding 下拉菜单中选择其他编码。点击 Save 保存文件

提　示　使用 .html 还是 .htm 并无区别，但推荐使用 .html，因为它更常见。无论你用哪种扩展名，请保持一致，因为使用相同的扩展名有助于后续记住 URL。

提示 即使已经指定了 .html 或 .htm 的扩展名，Windows 上的某些文本编辑器也会在文件名末尾加上默认扩展名。（注意大多数专门编辑 HTML 页面的编辑器并不存在这个问题。）这样文件名就变成了 webpage.html.txt，这样的文件无法在浏览器中查看。Windows 通常会隐藏扩展名，也导致这个问题变得隐蔽，它尤其容易困扰新手。有两个解决办法：一个是在首次保存文件时将文件名包围在双引号中，这样能防止添加额外的扩展名；另一个是让 Windows 显示文件扩展名（参见图 2.3.5 和图 2.3.6），从而可以看见程序自动添加的扩展名并将其删除。

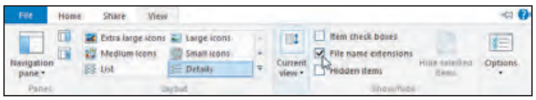


图 2.3.5 不同的 Windows 版本中菜单会有所不同（参见图 2.3.6）。在 Windows 8 中，可以从 Windows 资源管理器中选择 View（查看）选项卡找到这个菜单。如果想在桌面上和文件夹里看到文件的扩展名（如 .html），就要选取 File name extensions（让复选框打上勾）

提示 选择将文件保存为纯文本格式时，文件会以系统默认字符编码保存。如果需要创建其他编码的网页，需要使用可以选择编码类型的文本编辑器。通常，UTF-8 是最好的选择。如果编辑器中可选择的文件编码类型包括 UTF-8, no BOM 或类似的选项，请选择该项。否则，就选择 UTF-8（参见图 2.3.7）。有时，编辑器的 UTF-8 模式并不包含 BOM，但编码类型选择菜单中并未显式指出这一点。（如果你对 BOM 的含义极感兴趣，可以参见 http://en.wikipedia.org/wiki/Byte_order_mark。做好看不懂的准备吧！）

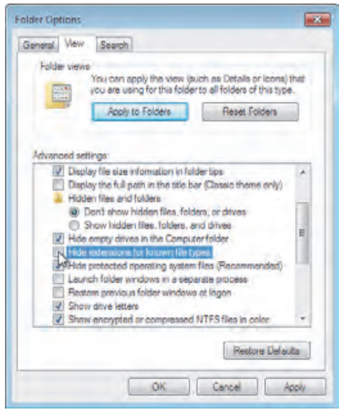


图 2.3.6 在 Windows 的早期版本中，这个菜单是上面这个样子的（根据 Windows 版本的不同，它们稍有差异）。这里是 Windows 7 中的菜单。在 Windows 资源管理器中，选择 Organize → Folder and search options（组织→文件夹和搜索选项），或者 Tools → Folder Options（工具→文件夹选项），具体取决于用户的 Windows 版本，显示图中的对话框。点击 View 选项卡，向下滚动，直到看到 Hide extensions for known file types（隐藏已知文件类型的扩展名）。如果想在桌面上和文件夹里看到文件的扩展名（如 .html），要确保这个选项是未选中的

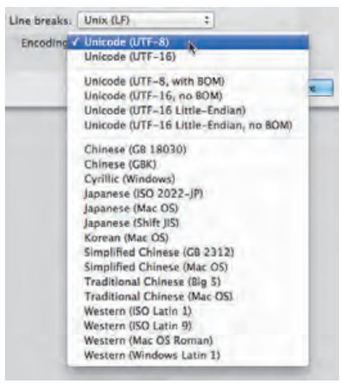


图 2.3.7 许多文本编辑器允许为文件选择编码，从而可以在同一个文档中保存不同语言中的符号和字符。大多数情况下，推荐选择 UTF-8。如果有 UTF-8, no BOM 的话就选这个选项，否则，直接选择 UTF-8。有的编辑器（如图中的 TextWrangler）会默认选择该项

2.4　指定默认页面或主页

大多数 Web 服务器都会根据文件名识别每个文件夹中的默认页面。大多数情况下，系统会将 index.html 识别为默认页面，参见图 2.4.1。如果没有 index.html，就会继续寻找 index.htm、default.htm 等文件名。如果访问者输入带目录的 URL，但没有指定文件名，那么就会打开默认页面，参见图 2.4.2。（目录就是文件夹，就像你自己计算机里的文件夹一样。）

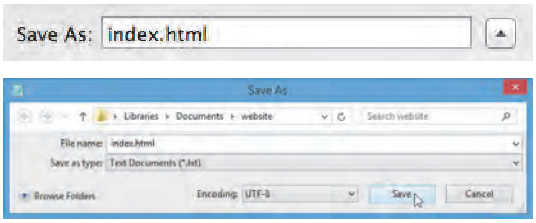


图 2.4.1　将文件保存为 index.html，从而指定这个文件是该目录下的默认页面（上图是 TextWrangler，下图是记事本）



图 2.4.2　当访问者输入目录的路径而没有指定文件名时，就会打开使用默认名称的页面。在这个例子中，输入的 URL 是 http://htmlcssvqs.com/gaudi/。如果输入 http://htmlcssvqs.com/gaudi/index.html，会显示相同的页面

在网站最顶层目录（通常称为根目录）中创建的默认页面是网站的主页。当访问者只输入域名而没有指定路径信息（如 www.*yourdomain*.com）时，就会显示这个页面。如果输入 www.*yourdomain*.com/index.html，结果是一样的（这里假定主页的文件名是 index.html）。

类似地，可以为网站的任何一个目录（甚至每一个目录）创建默认页面。例如，网站中 /products/ 或 /about-us/ 目录的着陆页（即首页）也应该是 index.html，只不过它们放在各自的目录里。网站的访问者通常通过主页或每页都有的主导航访问这些页面。

指定网站主页或含目录的着陆页的方法

在目标文件夹中，将文件保存为 index.html。（当你将文件上传到网站的服务器以后，如果 index.html 无法起到默认页面的作用，请向 Web 主机提供商咨询。关于将网页上传到服务器，参见第 21 章。）

当默认页面不存在时

如果目录中没有默认页面，有的服务器就会将目录文件列表显示出来（你可能并不希望向访问者暴露这些内容）。为了防止这种情况的发生，应该在网站每一个包含 HTML 页面的目录下创建一个默认页面。另一种办法是修改服务器的配置，将文件列表隐藏起来（如果它们已被隐藏，也可以将它们显示出来）。对于包含图像、媒体文件、样式表以及 JavaScript 等资源的目录，建议隐藏文件列表。如果不知道如何修改服务器配置，请向 Web 主机提供商咨询。

2.5　编辑网页

因为网页在大多数情况下是通过浏览器查看的，所以在桌面上双击网页文件，会启动默认浏览器并显示它们。如果想编辑网页

文件，需要在文本编辑器中手动打开它。

编辑网页的步骤

(1) 打开文本编辑器。

(2) 选择 File → Open。

(3) 找到包含目标文件的目录。

(4) 如果没有看到目标文件，选择 All Files（全部文件）选项（或类似的叫法），如图 2.5.1 和图 2.5.2 所示。在不同的程序或平台，这个选项的叫法和位置都有细微的差异，如图 2.5.3 所示。

(5) 点击 Open，就可以开始编辑文件了。

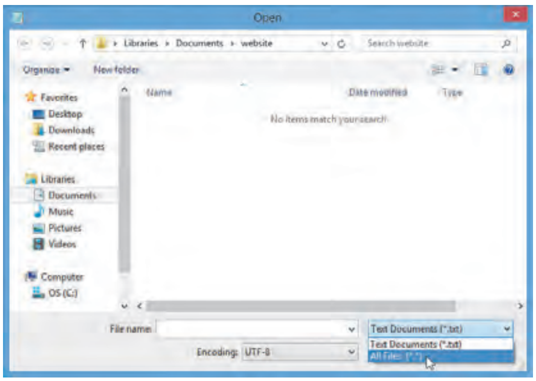


图 2.5.1　在 Windows 的一些文本编辑器（如记事本）中，不会自动看到 HTML 文件。选择 All Files 可以查看所有文件

提 示　可以从文件所在的文件夹直接打开文件（如果文件在桌面上，也可以直接从桌面上打开）。

提 示　通常，要保存对已有文档进行的修改，只需要选择 File → Save 即可，不必担心 2.3 节中提到的格式问题。使用快捷键保存文件更快。通常，编辑器中为保存设置的快捷键是 Command-S（对于 OS X）或 Ctrl-S（对于 Windows）。

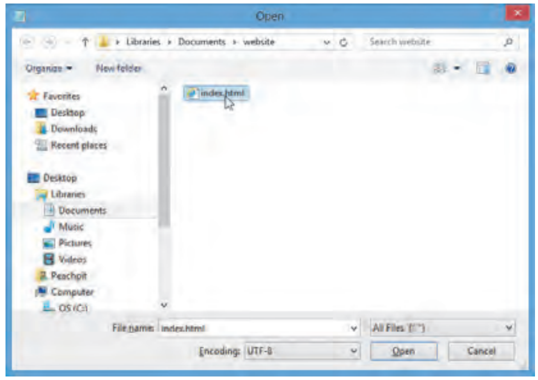


图 2.5.2　显示所有文件以后选中要编辑的 HTML 文件，点击 Open

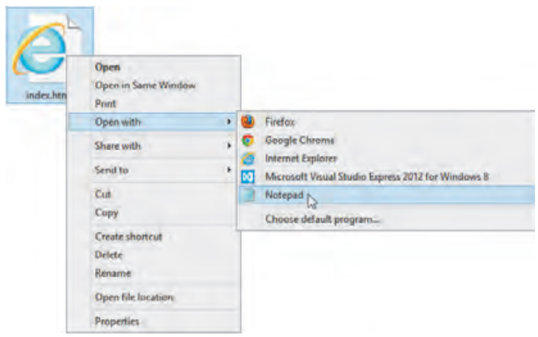


图 2.5.3　在 Windows 中，还可以右击文档的图标，在弹出菜单中选择 Edit 或 Open。在 Mac 上，右击图标或文件名，在弹出菜单中选择 Open With，再选择要使用的文本编辑器。在两种系统中，如果编辑器之前处于关闭状态，现在都会启动

2.6　组织文件

在文件数量变得很大之前，最好考虑好将它们放在什么地方。通常（但非必须）为网站的主要区块创建单独的文件夹，将相关的 HTML 页面放在一起。

组织文件的步骤

(1) 创建一个主文件夹或目录存放网站上所有可用的资料。在 Mac 上，在 Finder 中选

择 File → New Folder（文件→新文件夹），如图 2.6.1 所示。在 Windows 中，右击桌面（或你选择的文件夹），选择 New → Folder（新建→文件夹），如图 2.6.2 所示。然后为文件夹命名。

(2) 根据网站的组织结构创建子文件夹，如图 2.6.1 和图 2.6.3 所示。例如，可以考虑为网站的每个部分创建单独的文件夹，并根据需要在其中创建单独的子文件夹。

(3) 为网站的图像、样式表（CSS 文件）和 JavaScript 文件创建一个或多个文件夹，每个文件夹还可以有各自的子文件夹。有很多种组织方式，具体做法完全取决于你自己。一种做法是像图 2.6.1 和图 2.6.3 中显示的那样进行组织，还有一种方法是将 CSS、JavaScript 文件夹同图像文件夹（及其他文件夹）一样放在根目录，也可以将这些文件夹一起放在根目录下的 assets（资源）文件夹里。

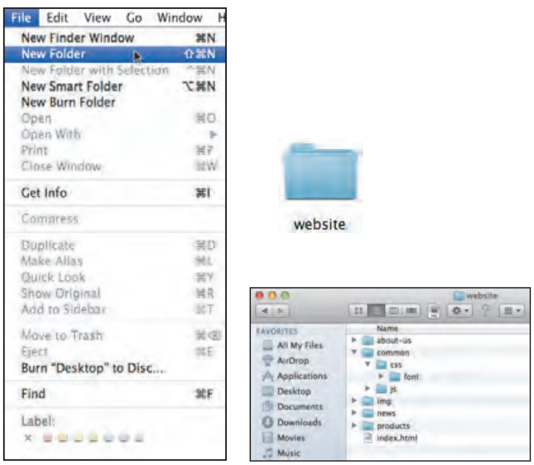


图 2.6.1　在 Mac 上，选择 New Folder，再为文件夹命名。为网站每个不同的部分创建独立的文件夹

提 示　第 (2) 步和第 (3) 步是可选的，但推荐这样做。

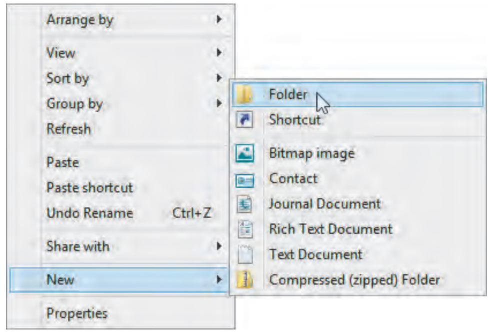


图 2.6.2　在 Windows 里，在桌面或资源管理器中右击，再选择 New → Folder

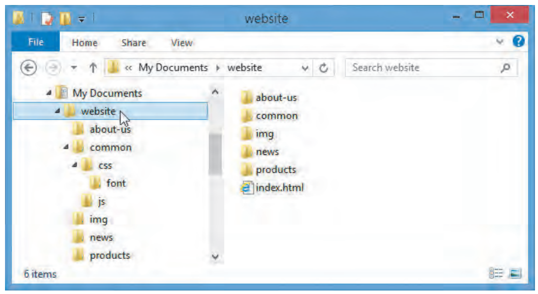


图 2.6.3　根据需要，可以将文件夹分解成多个子文件夹

提 示　使用短的、描述性的名称作为文件和文件夹的名称，使用短横线（而不用空格）分隔不同的单词。全部使用小写字母可确保 URL 更容易输入，页面更容易被访问到。关于创建文件名的细节，参见 1.6 节。

2.7　在浏览器中查看网页

创建网页之后，你肯定希望看看它在浏览器中显示的样子。实际上，你并不知道访问者使用什么浏览器（不同的浏览器呈现页面的方式会有差异），因此建议使用多个浏览器查看页面（参见 20.2 节）。

在浏览器中查看网页的步骤

(1) 打开浏览器。

(2) 选择 File → Open File（在不同的浏览器中，Open File 项的表述可能不同，但不管是什么，不能选择 Open Location），参见图 2.7.1。也可以使用快捷键 Command-O（对于 OS X）或 Ctrl-O（对于 Windows）。使用快捷键会直接跳到第 (3) 步，但 Internet Explorer 除外——IE 会先弹出一个如图 2.7.1 所示的对话框。

(3) 在弹出的对话框中，找到目标文件所在的文件夹，选中该文件，点击 Open（参见图 2.7.2）。页面会显示在浏览器中（参见图 2.7.3），显示效果与发布到服务器的网页相同（关于发布网页，参见第 21 章）。对于不同的浏览器，这些步骤可能有些差异。

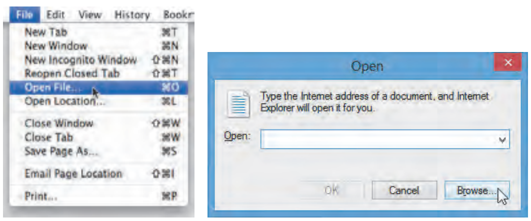


图 2.7.1　在 OS X 上，从浏览器中选择 File → Open File（上图为 Chrome）。在 Internet Explorer 中使用 Ctrl-O，在 Open（打开）对话框中点击 Browse（浏览）才能跳到下一步（下图）

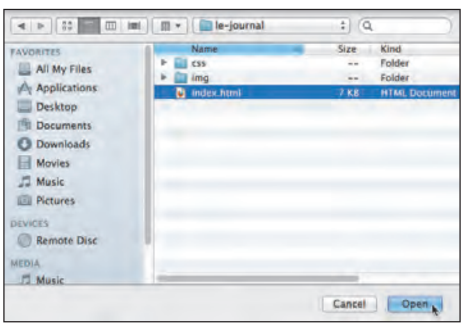


图 2.7.2　选择要打开的文件并点击 Open 按钮（这里显示的是 OS X 上的对话框，不过 Windows 上的是相似的）

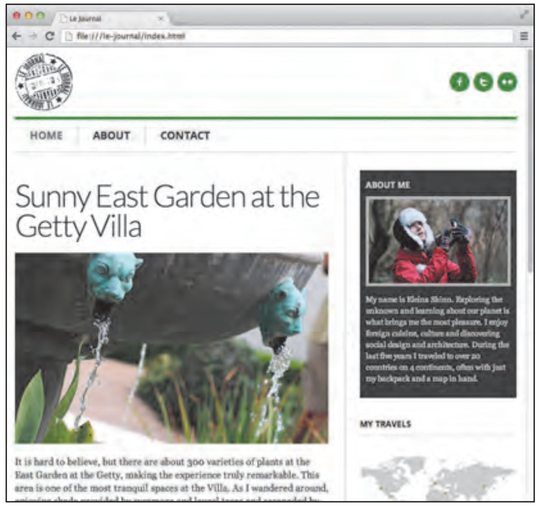


图 2.7.3　页面出现在浏览器中。仔细检查一下，看看它是否符合预期

提 示　通常还可以双击网页文件名或图标来查看该网页。如果你已经打开了浏览器，还可以将网页图标拖到浏览器窗口，这样也能打开。一旦掌握这种方法，它往往是在浏览器中查看网页的最简单方法。

提 示　一些现代浏览器没有 File → Open 这样的菜单选项来打开网页，试试第 (2) 步中提到的快捷键，或者使用上一条提示中提到的拖曳方法。

提 示　如果网页没有出现在选取文件的对话框中（图 2.7.2），那么检查一下文件是否以纯文本格式保存，且以 .html 或 .htm 为扩展名（参见 2.3 节）。

提 示　在浏览器中查看网页之前，不需要在文本编辑器中关闭文档，但一定要保存文件。在浏览器中打开网页之后，如果又在文本编辑器中对页面作了修改，那么需要再次保存

文件，并在浏览器中刷新页面。执行刷新操作最好使用浏览器的快捷键：Command-R（OS X）或 Ctrl-F5（Windows）。使用这些快捷键刷新网页可以避免浏览器使用缓存的网页（对于 Chrome，参见补充材料）。

提示　只有在 Web 服务器上发布网站（参见第 21 章）之后，访问者才能看到网站。

禁用 Chrome 缓存

Chrome 的缓存机制设计得有些特别。即便使用快捷键 Command-R（对于 OS X）或 Ctrl-F5（对于 Windows）刷新网页，有时还是会显示缓存的页面，而不是修改保存过的最新文件。查看网页的时候出现这种情况会产生误导，也可能让人烦躁。不过，幸好可以采取一些方法禁用 Chrome 缓存，这样就能总是显示最新的网页，方法如下。

(1) 在 Chrome 中按下 Command-Option-I（对于 OS X）或 Ctrl-Shift-I（对于 Windows），打开 Chrome 的 Developer Tools（开发者工具）。

(2) 点击右下角的齿轮图标。

(3) 在 Settings（设置）→ General（通用）中，选择 Disable Cache（禁用缓存）。

(4) 关闭 Settings 面板，但不关闭 Developer Tools。只要 Developer Tools 是打开状态的，就会禁用缓存（除非你取消选择 Disable Cache）。

顺便提一下，要让 Developer Tools 脱离 Chrome 窗口，显示为独立的窗口，可以点击 Developer Tools 左下角的图标。

2.8　借鉴他人灵感

学习其他网页开发和设计人员如何创建页面是提高 HTML 代码水平最容易的方法之一。幸好，我们很容易就可以查看和学习他人的 HTML 代码。不过，文本内容、图像、音频、视频、样式表及其他外部文件可能受版权保护。通常的做法是借鉴其他人的页面为自己的 HTML 寻找灵感，再创建自己的内容。

1. 使用 View Source 查看其他设计者的 HTML 代码

(1) 在浏览器中打开网页。

(2) 选择 View Source（查看源代码，或浏览器中的其他类似选项），如图 2.8.1 和图 2.8.2 所示。这时会显示 HTML 代码，如图 2.8.3 所示。

(3) 如果需要，可以保存该文件以作进一步研究。

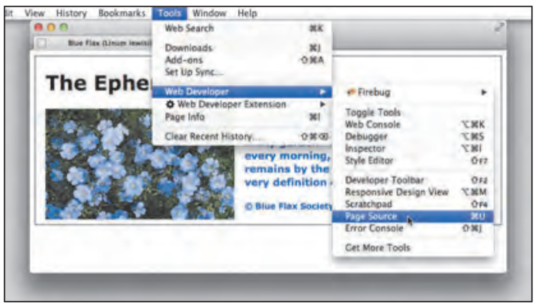


图 2.8.1　所有的桌面浏览器都有让你查看页面 HTML 源代码的菜单选项。选项的名称可能是 View Source，也可能是 Page Source（图中的 Firefox），或者其他类似的名称。在 Chrome 中为 Tools → View Page Source（工具→查看页面源代码）

2. 通过开发者工具查看其他设计者的 HTML 代码

查看网页源代码的另一种方法是使用浏览器的开发者工具。不同浏览器提供的开发者工具并不一样，但有些功能是一样的。

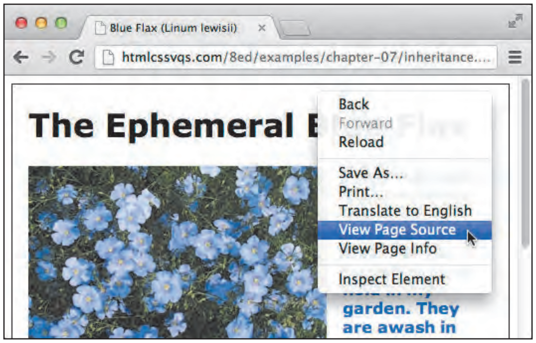


图 2.8.2　大多数浏览器还可以在页面上右击，然后在弹出的菜单中选择 View Source（或者别的类似叫法）。图上显示的是 Chrome 中的菜单。这通常是查看源代码最方便的方法，因为很难在主菜单或子菜单上找到该选项



图 2.8.3　现代浏览器在其自身的选项卡或窗口中显示源代码（如图所示），而旧的浏览器可能在特定的文本编辑器中显示。页面内容的颜色与 HTML 元素、属性和属性值的颜色是区分开来的。这称为语法高亮（syntax highlighting）。左侧显示的行号并不是 HTML 代码的一部分，也并非所有的浏览器都会在 View Source 模式中显示行号

这些工具提供了一种交互式的查看源代码的视图。可以审查页面特定部分的 HTML 和 CSS，在浏览器中编辑代码，并立即查看修改后的效果。开发者工具适用任何网站（不仅仅是自己的）。使用开发者工具作的修改都是临时的，这些工具并未真正修改页面的 HTML 和 CSS。开发者工具是学习 HTML 和 CSS 不可多得的资料，通过它可以查看某个特定的效果是如何实现的，可以随意改动代码而不必担心破坏任何东西。

关于开发者工具（既有现代浏览器的，也有旧浏览器的）的信息，参见 20.1 节中的“浏览器开发者工具”。

提 示　不存在关于谁能在万维网上发布网站的规则。这就是万维网的伟大之处——它是一种进入门槛很低的开放媒体。你可以是一名新手，一位专家，或者介于二者之间的角色。查看其他网站源代码时应该记住这一点，如果有些代码看起来并不怎么样，不要仅仅因为该网站放在网上就认为其制作者知道的比你多。有大量网站可以作为编码最佳实践的范例，也有大量网站做得并非那么理想。保持批判的态度，如果对某项技术的合理性有所怀疑，请查看本书或其他资源。

提 示　要保存代码，可以将代码从 View Source 窗口中复制到文本编辑器中，再保存文件。在 OS X 上，使用 Command-A 选择所有代码，再使用 Command-C 复制代码。再切换到编辑器中，按下 Command-V 粘贴代码。在 Windows 上，过程是相似的，只是用 Ctrl 键代替 Command 键。

提 示　要同时保存源代码及网页资源（如图像），在大多数浏览器中可以选择 File → Save As（或 File → Save Page As，文件 → 页面另存为）。不过，保存页面时，浏览器可能改写代码的某些部分，因此代码与使用上一条提示保存的代码并不完全一样。

提 示　关于查看网页的 CSS，参见 8.7 节。

第 3 章

基本 HTML 结构

本章内容
- ❑ 开始编写网页
- ❑ 创建页面标题
- ❑ 创建分级标题
- ❑ 普通页面构成
- ❑ 创建页眉
- ❑ 标记导航
- ❑ 标记网页主要区域
- ❑ 创建文章
- ❑ 定义区块
- ❑ 指定附注栏
- ❑ 创建页脚
- ❑ 创建通用容器
- ❑ 使用 ARIA 改进可访问性
- ❑ 为元素指定类或 ID
- ❑ 为元素添加 title 属性
- ❑ 添加注释

本章讨论的是构建文档基础和结构所需的HTML元素，即网页内容主要的语义化容器。

你将学到以下内容：
- ❑ 开始编写网页；
- ❑ HTML5 的文档大纲；
- ❑ h1~h6、header、nav、main、article、section、aside、footer 及 div 元素（其中大多数是 HTML5 新增的元素）；
- ❑ 使用 ARIA 的 role 属性提升页面的可访问性；
- ❑ 为元素应用类或 ID；
- ❑ 为元素应用 title 属性；
- ❑ 为代码添加注释。

创建清晰、一致的结构不仅可以为页面建立良好的语义化基础，也可以大大降低在文档中应用层叠样式表（CSS）的难度（从第 7 章开始讲解 CSS）。

3.1　开始编写网页

每个 HTML 文档都应该包含以下基本成分（如图 3.1.1 所示）。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title></title>
</head>
<body>



</body>
</html>
```

图 3.1.1　这是每个 HTML 页面的基础。缩进并不重要，但结构很重要。在这个例子中，默认语言（由 lang 属性指定）被设为代表英语的 en。字符编码被设为 UTF-8

- ❑ DOCTYPE

- html 元素（包含 lang 属性，该属性不是必需的，但推荐加上）
- head 元素
- 说明字符编码的 meta 元素
- title 元素
- body 元素

这份 HTML 等同于一张空白的纸，因为 body 里面没有任何内容，如图 3.1.2 所示。

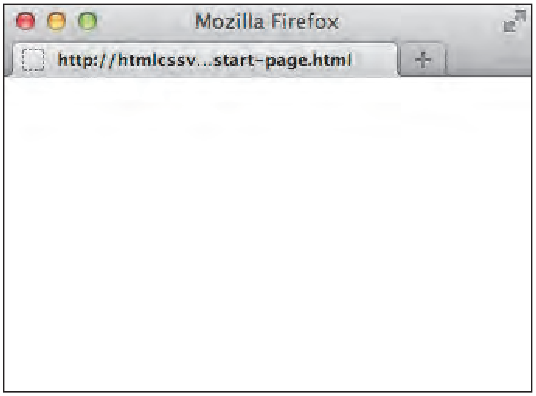


图 3.1.2　最少 HTML 基础代码在 Firefox 中显示的样子。如你所见，什么内容也没有！不过，很快就可以开始添加内容了

在添加任何内容或其他信息之前，需要先建立起页面的基础。

1. 编写 HTML5 页面开头的步骤

(1) 输入 <!DOCTYPE html>，声明页面为 HTML5 文档。（关于 HTML 早期版本的相关信息，参见本节末尾的“改进后的 HTML5 DOCTYPE”。）

(2) 输入 <html lang="*language-code*">，开始文档的实际 HTML 部分。其中，*language-code* 是页面内容默认语言的代码。例如，<html lang="es"> 表示西班牙，<html lang="fr"> 表示法语。还可以写得更详细些，如 <html lang="en-US"> 表示美国英语，而 <html lang="en-GB"> 则表示英国英语。

(3) 输入 <head>，开始网页文档的头部。

(4) 输入 <meta charset="utf-8"/>（或 <meta charset="UTF-8"/>），将文档的字符编码声明为 UTF-8。空格和斜杠是可选的，因此 <meta charset="utf-8"> 跟其他表达式形式都是有效的。（UTF-8 以外的其他字符编码也是有效的，不过 UTF-8 的适用面最广，很少有需要用其他编码的情况。）

(5) 输入 <title></title>。这里将包含页面的标题。3.2 节中将讲解如何添加标题文字。

(6) 输入 </head>，结束页面文档的头部。

(7) 输入 <body>，开始页面的主体。这里是放置页面内容的地方。

(8) 为页面内容预留一些空行。后续章节会讲解如何创建内容。

(9) 输入 </body>，结束主体。

(10) 输入 </html>，结束页面。

步骤似乎有点多，不过，由于所有的页面都是这样开始编写的，可以使用一个单独的 HTML 页面作为创建每个页面的模板，以节省输入的时间。实际上，很多代码编辑器都可以为新页面指定初始的代码，这样就更简单了。如果你在编辑器中没有找到 Settings（设置）或 Preferences（偏好设置）菜单，可以在 Help（帮助）中搜索。

2. 网页的两个部分：head 和 body

快速回顾一下第 1 章学到的知识，HTML 页面分为两个部分：head 和 body，如图 3.1.1 所示。DOCTYPE 出现在每个页面的开头，就像一本书的序言。

在文档 head 部分，通常要指明页面标题，提供为搜索引擎准备的关于页面本身的信息，加载样式表，以及加载 JavaScript 文件（不过，出于性能考虑，多数时候在页面底部 </body> 标签结束前加载 JavaScript 是更好的选择）。随着进一步阅读本书，你会看到一些关于这

些内容的例子。除了 title，其他 head 里的内容对页面访问者来说都是不可见的。

body 元素包住页面的内容，包括文本、图像、表单、音频、视频以及其他交互式内容，也就是访问者看见的东西。本书有好几章都讲述 HTML 的内容相关元素，本章会提前接触到其中的一些。

提示 使用 HTML5 DOCTYPE 可以确保浏览器以可靠的模式呈现页面，同时告诉 HTML 验证器根据 HTML5 允许的元素和语法检查代码。第 20 章会讨论 HTML 验证器。

提示 HTML5 的 DOCTYPE 不区分大小写。有的人可能输入 <!doctype html>，不过，使用 <!DOCTYPE html> 是更常规的做法，参见图 3.1.1。

提示 搜索引擎可能会根据 lang 属性指定的语言区分搜索结果，从而仅显示与搜索词语言相同的页面。屏幕阅读器也可能通过指定的语言调整其发音。

提示 要查找语言代码，可以使用 Richard Ishida 的语言标签查询工具（http://rishida.net/utils/subtags/）。

提示 要确保将你的代码编辑器配置为以 UTF-8 保存文件，与代码中通过 <meta charset="utf-8" /> 语句指定的编码相同，参见图 3.1.1。（如果指定了另一种编码，就按那种编码保存文件。）并非所有的编辑器都默认以 UTF-8 保存文件，但大多数编辑器可以在菜单或面板中选择编码（参见 2.3 节）。如果页面没有设置为 UTF-8，有的字母（如带重音符的 i、带波形符（～）的 n）就会变成一些奇怪的字符。

提示 嵌套在 head 元素里的代码不一定要缩进，参见图 3.1.1。不过，缩进可以让你一眼看出 head 从哪里开始，包含什么内容，以及在哪里结束。在有些页面中，head 变得很长并不少见。

改进后的 HTML5 DOCTYPE

有了 HTML5 以后，编写代码的开头部分变得容易多了。特别是跟从前的 DOCTYPE 相比，HTML5 的 DOCTYPE 短得让人惊讶。

在 HTML 4 和 XHTML 1.0 时代，有好几种可供选择的 DOCTYPE，每一种都会指明 HTML 的版本，以及使用的是过渡型还是严格型模式。由于它们太难记，开发人员不得不每次都从某个地方把这些代码复制进来。

例如，下面是 XHTML 严格型文档的 DOCTYPE。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
→ "http://www.w3.org/ TR/xhtml1/DTD/xhtml1-strict.dtd">
```

看起来令人费解。

幸好，所有的浏览器，无论新旧，都理解 HTML5 的 DOCTYPE，因此你可以坚持在所有页面中使用它，忘掉过去的那些 DOCTYPE。（唯一可能用到过去那些 DOCTYPE 的情况是继承一个旧站，而网站的拥有者不允许将 DOCTYPE 修改为 HTML5 的版本。）

3.2　创建页面标题

3.1 节 HTML 基础代码中 <title></title> 仅为占位符，现在开始讨论 title 元素。

每个 HTML 页面都必须有一个 title 元素。每个页面的标题都应该是简短的、描述性的，而且是唯一的，如图 3.2.1 所示。在大多数浏览器中，页面标题出现在窗口的标题栏（Chrome 是个例外）和浏览器的标签页中，如图 3.2.2 所示。页面标题还会出现在访问者浏览历史列表和书签里，如图 3.2.3 所示。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Antoni Gaudí - Introduction
    → </title>
</head>
<body>
</body>
</html>
```

图 3.2.1　title 元素必须位于 head 部分，将它放置在指定字符编码的 meta 元素后面

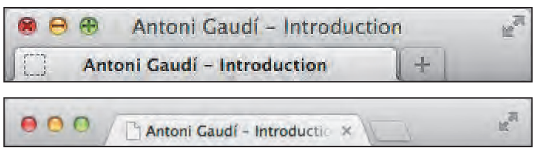


图 3.2.2　在大多数浏览器（如 Firefox）中，页面标题既显示在标题栏，也显示在标签页上。不过，Chrome（下图）只在标签页上显示页面标题

或许更为重要的是，页面标题会被 Google、Bing、DuckDuckGo、Yahoo! 等搜索引擎采用，从而能够大致了解页面内容，并将页面标题作为搜索结果中的链接显示，如图 3.2.4 所示。

总之，要让每个页面的 title 是唯一的，从而提升搜索引擎结果排名，并让访问者获得更好的体验。

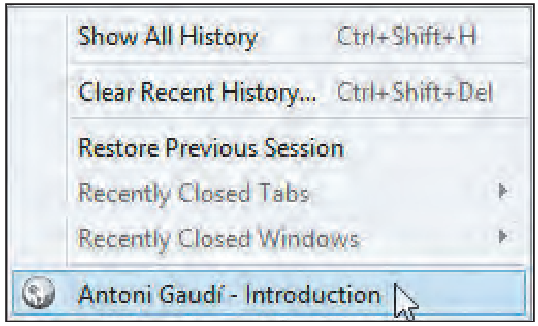


图 3.2.3　页面标题也出现在访问者的 History 面板（如图）、收藏夹列表以及书签列表中



图 3.2.4　或者最重要的是，页面标题通常是 Google 等搜索引擎的搜索结果中链接的文字，它也是判断搜索结果中页面相关度的重要因素。此处为 Google 中显示的页面标题和部分主体内容

创建页面标题的步骤

(1) 将光标放在文档 head 中的 <title> 和 </title> 之间。

(2) 输入网页的标题。

提 示　title 元素是必需的。

提 示　title 中不能包含任何格式、HTML、图像或指向其他页面的链接。

提 示　创建新页面时，有的代码编辑器会预先为页面标题填上默认文字，除非已经按照 3.1 节中介绍的方法使用了特定的开头代码。要注意到这些默认文字，确保用自己的标题替换它们。

深入探讨页面标题

很多开发人员不太重视 title 元素，甚至那些相当用心、很有经验的开发人员也是这样。他们只是简单地输入网站名称，并将其复制到全站每一个网页。或者更糟糕，让 title 文字仍然保存为代码编辑器默认添加的文字。如果流量是网站追求的指标之一，这样做对建站者和潜在的访问者都会产生巨大的损害。

不同搜索引擎确定网页排名和内容索引规则的算法是不一样的。不过，title 通常都扮演着重要的角色。搜索引擎会将 title 作为判断页面主要内容的指标，并将页面内容按照与之相关的文字进行索引。有效的 title 应包含几个与页面内容密切相关的关键字。

作为一种最佳实践，选择能简要概括文档内容的文字作为 title 文字。这些文字既要对屏幕阅读器用户友好，又要有利于搜索引擎排名。

其次，将网站名称放入 title（这不是必需的）。常见的做法是将网站名称放在 title 的开头，不过将页面特有的文字放在开头更好。

建议将 title 的核心内容放在前 60 个字符（含空格）中，因为搜索引擎通常将超过此数目（作为基准）的字符截断。不同浏览器显示在标题栏中的字符数上限不尽相同。浏览器标签页会将标题截得更短，因为它占的空间较少。

3.3　创建分级标题

HTML 提供了六级标题用于创建页面信息的层级关系。使用 h1、h2、h3、h4、h5 或 h6 元素对各级标题进行标记，其中 h1 是最高级别的标题，h2 是 h1 的子标题，h3 是 h2 的子标题，以此类推。为简洁起见，本书使用 h1 ~ h6 表示这些元素，不再逐一列出。

为了理解 h1 ~ h6 标题，可以将它们比作学期论文、销售报告、新闻报道、产品手册等非 HTML 文档里的标题。当你撰写这些文章时，会根据需要为内容的每个主要部分指定一个标题和任意数量的子标题（以及子子标题，等等）。总之，这些标题代表了文档的大纲。网页的分级标题也是这样的（参见图 3.3.1 和图 3.3.2）。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Antoni Gaudí - Introduction
    ➝ </title>
</head>
<body>
<h1>Barcelona's Architect</h1>
<h2 lang="es">La Sagrada Família</h2>
<h2>Park Guell</h2>
</body>
</html>
```

图 3.3.1　用标题建立文档结构，就像大纲一样。这里，标记为 h2 的 La Sagrada Família 和 Park Guell 是标记为 h1 的顶级标题 Barcelona’s Architect 的子标题。如果 Park Guell 是 h3，它就成了 La Sagrada Família 的子标题（也是 h1 的子子标题）。如果继续编写页面其余部分的代码，相关的内容（段落、图像、视频等）就要紧跟在对应的标题后面

1. 分级标题的重要性

对任何页面来说，分级标题都可以说是最重要的 HTML 元素。由于标题通常传达的是页面的主题，因此，对搜索引擎而言，如果标题与搜索词匹配，这些标题就会被赋予很高的权重，尤其是等级最高的 h1（这并不是说页面中的 h1 越多越好，搜索引擎还是足够聪明的）。

```
...
<body>
<h1>Product User Guide</h1>
    <h2>Setting it up</h2>

    <h2>Basic Features</h2>
       <h3>Video Playback</h3>
          <h4>Basic Controls</h4>
          <h4>Jumping to Markers</h4>

       <h3>Recording Video</h3>
          <h4>Manual Recording</h4>
          <h4>Scheduling a Recording</h4>

    <h2>Advanced Features</h2>
       <h3>Sharing Video</h3>
       <h3>Compressing Video</h3>
</body>
</html>
```

图 3.3.2 在这个例子中，产品指南有三个主要的部分，每个部分都有不同层级的子标题。标题之间的空格和缩进只是为了让层级关系更清楚一些，它们不会影响最终的显示效果。在实践中我通常不会添加这样的空格和缩进，如果你想添加可以这样做

对人来说，好的分级标题也很重要。视力良好的用户需要通过分级标题确定页面内容。屏幕阅读器用户也是这样，只不过是通过手和耳朵。他们通常通过键盘按键在标题间移动，这样可以快速了解页面内容并去查看最感兴趣的内容，而不用把整个页面从头到尾听完。当他们找到感兴趣的标题之后，可以选择阅读该标题下的内容。可见 h1 ~ h6 对可用性和无障碍访问的作用是很大的。

总之，好的标题层级结构对你和你的访客来说都很重要。

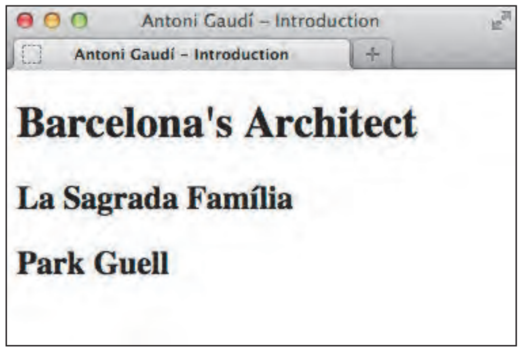
2. 使用标题对网页进行组织的步骤

(1) 在 HTML 的 body 部分，输入 <h*n*>，其中 *n* 是 1 ~ 6 的数字，此数字取决于要创建的标题的级别。h1 是最重要的标题，而 h6 则是最不重要的标题。

(2) 输入标题的内容。

(3) 输入 </h*n*>，其中 <*n*> 是与第 (1) 步中相同的数字。

提 示 在默认情况下，浏览器会从 h1 到 h6 逐级减小标题的字号，如图 3.3.3 所示。（在有的浏览器里，嵌套在特定元素中的 h1 和 h2 在默认情况下看起来是一样的。）不过别忘了要依据内容所处的层次关系选择标题级数，而不是根据你希望文字应该显示的大小。可以按照你希望的样子为标题添加样式，包括字体、字号、颜色等。用 CSS 实现这些样式的详细说明参见第 10 章。



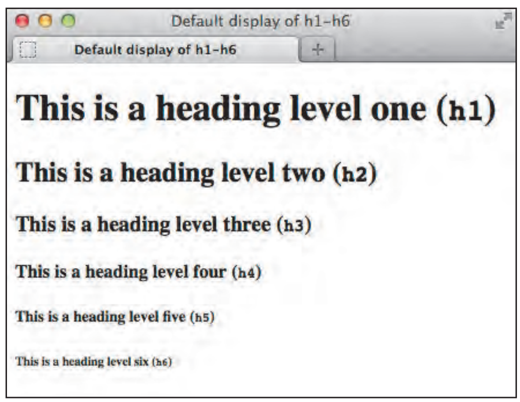


图 3.3.3 在默认情况下，所有的标题都以粗体显示，h1 的字号比 h2 的大，而 h2 的又比 h3 的大，以此类推。每个标题之间的间隔也是由浏览器默认的 CSS 制造的，它们并不代表 HTML 文档中有空行

提示 创建分级标题时，要避免跳过某些级别，如从 h3 直接跳到 h5。不过，允许从低级别跳到高级别的标题。例如，在图 3.3.2 中，<h4>Scheduling a Recording</h4> 后面紧跟着 <h2>Advanced Features</h2> 是没有问题的，因为包含 Scheduling a Recording 的 Basic Features（也是 h2）在这里结束了，而 Advanced Features 的内容开始了。

提示 不要使用 h1 ~ h6 标记副标题、标语以及无法成为独立标题的子标题。例如，假设有一篇新闻报道，它的主标题后面紧跟着一个副标题，这时，这个副标题就应该使用段落（参见图 3.3.4）或其他非标题元素。

提示 先前，HTML5 包含了一个名为 hgroup 的元素，用于将连续的标题组合在一起，W3C 将这个元素从 HTML5 规范中移除了。

```
…
    <h1>Giraffe Escapes from Zoo</h1>
    <p class="subhead">Animals Worldwide
    ➝ Rejoice</p>

    <p>… story content …</p>
…
```

图 3.3.4　这是标记文章副标题的一种方法。可以添加一个 class，从而能够应用相应的 CSS。该 class 可以命名为 subhead 等名称

提示 另外注意一下，图 3.3.1 中对一个 h2 使用了 lang 属性，用以表明这里的文字为页面默认语言（英语，因为 <html lang="en">）之外的另一种语言（西班牙语，由 es 表示）。

提示 在本书写作之际，曾有人提议在 HTML 中引入 subhead 元素，用于对子标题、副标题、标语、署名等内容进行标记。目前还无从知道这个提议是否会被采纳。

HTML5 的文档大纲

在本书写作之际，HTML5 对如何处理位于 article、aside、nav 和 section 元素中的 h1 ~ h6 有一套算法。该算法通常称为 **HTML5 文档大纲**（HTML5 document outline）。不过，目前还没有浏览器实现这套算法，而且也看不到支持的迹象。而且，屏幕阅读器中只有 JAWS（一款运行于 Windows 下的屏幕阅读器）支持，而它的实现还存在问题。

鉴于此，W3C 已经将文档大纲列入可能从 2014 年最终定稿的规范中移除的特性。即便文档大纲最终留在规范中，或者浏览器将其实现了，我们还是可以按照这里介绍的方法对分级标题进行标记。这种方法不仅适用于当前环境，还足以应对未来（因为文档大纲不会对页面造成破坏）。

由此看来，你大可不必理会文档大纲。这里提到文档大纲只是为了让你在其他地方看到时不必犹豫是否需要了解其机理。

3.4　普通页面构成

你肯定已经访问过了大量像图 3.4.1 中显示的那种网站。抛开内容不谈，我们可以看到该页面有四个主要组件：带导航的页头、显示在主体内容区域的文章、显示次要信息的附注栏以及页脚，如图 3.4.2 所示。



图 3.4.1　一个普通的布局，顶部是主导航，左侧是主要内容，右侧是附注栏，底部是页脚。要让页面成为这个样子，需要添加 CSS

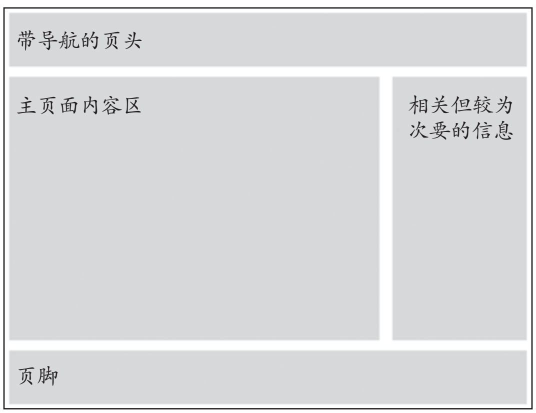


图 3.4.2　页面的常规信息类型。这只是一种安排方式，不过是很常见的一种

现在，在没有引入 CSS 的情况下，你还无法像这样为页面添加样式。我们将从第 7 章开始学习 CSS，在第 10 章开始学习如何对文字进行格式化、添加颜色等，在第 11 章学习多栏布局。

不过，应用到这些常规页面结构的语义都是非常相似的，与布局无关。本章剩余章节的大部分内容都会讲解这些结构。按照从页面顶端向下的顺序，将依次讲解用 `header`、`nav`、`main`、`article`、`section`、`aside` 和 `footer` 定义页面的结构，以及用来添加额外样式信息或实现其他目的的通用容器 `div`。除了 `div` 以外，这些元素都是 HTML5 推出后才有的。

在学习这些元素的过程中，不要关心它们在示例布局中的位置，而应该关注它们的语义。

接下来，我们还会先大概了解一下其他元素，如 `ul`（无序列表）和 `a`（链接），后续章节会详细讲解。

3.5　创建页眉

如果页面中有一块包含一组介绍性或导航性内容的区域，应该用 `header` 元素对其进行标记。

一个页面可以有任意数量的 `header` 元素，它们的含义可以根据其上下文而有所不同。例如，处于页面顶端或接近这个位置的 `header` 可能代表整个页面的页眉（有时称为页头），如图 3.5.1 所示。通常，页眉包括网站标志、主导航（图 3.5.2）和其他全站链接，甚至搜索框（图 3.5.3）。这无疑是 `header` 元素最常见的使用形式，不过不要误认为是唯一的形式。

```
...
<body>
<header role="banner">
    <nav>
        <ul>
            <li><a href="#gaudi">Barcelona's
            → Architect</a></li>
            <li lang="es"><a href="#sagrada-
            → familia">La Sagrada Família</a>
            → </li>
            <li><a href="#park-guell">Park
            → Guell</a></li>
        </ul>
    </nav>
</header>
</body>
</html>
```

图 3.5.1　这个 header 代表整个页面的页眉。它包含一组代表整个页面主导航的链接（在 nav 元素中）。可选的 role="banner" 并不适用于所有的页眉。它显式地指出该页眉为页面级的页眉，因此可以提高可访问性。图 3.5.4 中也显示了一个这样的例子。要了解更多信息，参见 3.13 节（对于 nav 元素的 role 属性值，参见 3.6 节）



图 3.5.2　包含导航的页面级页眉

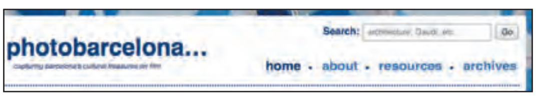


图 3.5.3　这是另一个网站的页眉（包含样式）。这种页面级页眉的形式在网上很常见。它包含网站名称（通常为一个标识）、指向网站主要板块的导航链接以及一个搜索框

header 也很适合对页面深处的一组介绍性或导航性内容进行标记。例如，一个区块的目录，参见图 3.5.4。

```
...
<body>
<header role="banner">
    ... [网站标识、导航等] ...
</header>

<main role="main">
<article>
    <header>
        <h1>Frequently Asked Questions</h1>
        <nav>
            <ul>
                <li><a href="#answer1">What is
                → your return policy?</a><li>
                <li><a href="#answer2">How do I
                → find a location?</a><li>
                ...
            </ul>
        </nav>
    </header>

    <!-- 第二个header的链接指向这里 -->
    <article id="answer1">
        <h2>What is your return policy?</h2>
        <p> ... [答案] ... </p>
    </article>

    <article id="answer2">
        <h2>How do I find a location?</h2>
        <p> ... [答案] ... </p>
    </article>
    ...
</article> <!-- 结束父元素article -->
</main>
</body>
</html>
```

图 3.5.4　这个页面有两个 header，一个是整个页面的，另一个是 Frequently Asked Questions 的父元素 article 的。注意第一个并不包含任何 h1 ～ h6 标题，而第二个则有。而且，只有第一个 header 中有 role="banner"，因为它是页面级的页眉

header 通常包含其自身的标题（h1 ～ h6），但这并不是强制性的。例如，图 3.5.3 中有标题，但图 3.5.1 中就没有。

创建页眉的步骤

(1) 将光标放置在需要创建页眉的元素里。

(2) 输入 <header>。

(3) 输入页眉的内容，包括各种类型的内容，它们分别由各自的 HTML 元素（本书余下部分将会讲到其中的大多数）进行标记。例如，header 可以包含 h1 ~ h6 标题、标识、导航、搜索框，等等。

(4) 输入 </header>。

提示 只在必要时使用 header。大多数情况下，如果使用 h1 ~ h6 能满足需求，就没有必要用 header 将它包起来。

提示 header 与 h1 ~ h6 元素中的标题（参见 3.3 节）是不能互换的。它们都有各自的语义目的。

提示 不能在 header 里嵌套 footer 元素或另一个 header，也不能在 footer 或 address 元素里嵌套 header。

提示 header 不一定要像示例那样包含一个 nav 元素（图 3.5.1 和图 3.5.4），不过在大多数情况下，如果 header 包含导航性链接，就可以用 nav。在图 3.5.4 所示的例子中，nav 包住 Frequently Asked Questions 链接列表是恰当的，因为它是页面内的主要导航组（将在 3.6 节中讨论）。

提示 要了解 header 如何取代 HTML5 之前版本中 div 元素的一个功能，参见 3.12 节。

3.6 标记导航

HTML 的早期版本没有元素明确表示主导航链接的区域，而 HTML5 则有这样一个元素，即 nav。nav 中的链接可以指向页面中的内容，如图 3.6.1 所示，也可以指向其他页面或资源，或者两者兼而有之。无论是哪种情况，应该仅对文档中重要的链接群使用 nav。

```
...
<body>
<header role="banner">
    <nav role="navigation">
        <ul>
            <li><a href="#gaudi">Barcelona's
            → Architect</a></li>
            <li lang="es"><a href="#sagrada-
            → familia">La Sagrada Família</a>
            → </li>
            <li><a href="#park-guell">Park
            → Guell</a></li>
        </ul>
    </nav>
</header>
</body>
</html>
```

图 3.6.1 这些链接（a 元素）代表一组重要的导航，因此将它们放入一个 nav 元素。role 属性并不是必需的，不过它可以提高可访问性。更多有关向 nav 应用 role="navigation" 的信息参见本节最后一个提示

如果你仔细看过上一节的代码，就已经见识过 nav 元素了。下面将那一段代码搬过来，并对 nav 进行了突出显示。nav 元素不会对其内容添加任何默认样式，如图 3.6.2 所示。



图 3.6.2 在默认情况下，我们的导航看起来相当普通。那些圆点并不是由 nav 元素产生的。除了开启一个新行以外，该元素没有任何默认样式。显示这些圆点是因为每个链接都包在一个 li 元素（列表项）里面。使用 CSS 可以隐藏这些圆点或显示其他的东西，还可以将这些链接水平显示，改变它们的颜色，让它们看起来像按钮，等等。本书将从第 7 章开始讲解 CSS

将一组链接指定为重要导航

(1) 输入 <nav>。

(2) 输入一组链接并将其标记为 ul（无序列表）结构，除非链接的顺序比较重要（例如面包屑导航）。如果链接顺序重要，就将其标记为 ol（有序列表）结构。（要了解链接和列表，分别参见第 6 章和第 15 章。）

(3) 输入 </nav>。

提示　有编写 HTML 或 XHTML 经历的开发人员或许已经习惯了使用 ul 或 ol 元素对链接进行结构化。在 HTML5 中，nav 并没有取代这种最佳实践。应该继续使用这些元素，只是在它们的外围简单地包上一个 nav（参见图 3.6.1）。

提示　尽管屏幕阅读器整体上还在追赶 HTML5 新出现的语义功能，但 nav 能帮助它们识别页面的主导航，并允许用户通过键盘直接跳至这些链接。这可以提高页面的可访问性，提升访问者的体验。

提示　HTML5 规范不推荐对辅助性的页脚链接（如“使用条款”、“隐私政策”等）使用 nav，这不难理解。不过，有时页脚会再次显示顶级全局导航，或者包含“商店位置”、“招聘信息”等重要链接。在大多数情况下，我们推荐将页脚中的此类链接放入 nav 中。

提示　HTML5 不允许将 nav 嵌套在 address 元素中。

提示　要了解如何在 nav 中使用 role="navigation"（参见图 3.6.1），请参见 3.13 节。

深入了解 nav

上文提到，要在页面中插入一组链接并非意味着一定要将它们包在 nav 元素里。

设想有一个如图 3.6.3 所示的新闻页面。这是一篇文章的 Arts & Entertainment 部分，是一则关于某画廊开业的短文。该页面包含四个链接列表，其中只有两个列表具有足够的重要性，可以包在 nav 中。（可以看到，这里省略了部分代码。）

位于 aside 中的次级导航允许用户跳转到 Arts & Entertainment 中的分类页面（如 Movies、Music），因此它构成了页面的一个主要导航区块。不过，Other Stories 这个 aside 则不是，footer 里的链接也不是。（关于 aside 元素，参见 3.10 节。）

那么，如何判断是否对一组链接使用 nav 呢？归根结底，这取决于内容的组织情况。至少，应该将网站全局导航（让用户可以跳至网站各个主要部分的导航）标记为 nav。这种 nav 通常（但并不总是）出现在页面级的 header 元素里面（参见 3.5 节）。

```
...
<body>
    <!-- ==== 开始页面级页眉 ==== -->
    <header role="banner">
        <!-- 站点标识可以放在这里 -->
        <!-- 全站导航 -->
        <nav role="navigation">
            <ul> ... </ul>
        </nav>
    </header>

    <!-- ==== 开始主要内容 ==== -->
    <main role="main">
        <h1>Arts & Entertainment: Museums
        ➝ </h1>

        <article>
            <h2>Gallery Opening Features the
            ➝ Inspired, Inspiring</h2>
            <p>... [故事内容] ... </p>
        </article>

        <aside>
            <h2>Other Stories</h2>
            <!-- 没有包含在nav里 -->
            <ul> ... [故事链接] ... </ul>
        </aside>
    </main>

    <!-- ==== 开始附注栏 ==== -->
    <aside>
        <!-- 次级导航 -->
        <nav role="navigation">
            <ul>
                <li><a href="/arts/movies/">
                ➝ Movies</a></li>
                <li><a href="/arts/music/">
                ➝ Music</a></li>
                ...
            </ul>
        </nav>
    </aside>

    <!-- ==== 开始页面级页脚 ==== -->
    <footer role="contentinfo">
        <!-- 辅助性链接并未包在nav中 -->
        <ul> ... </ul>
    </footer>
</body>
</html>
```

图 3.6.3 在这个页面中，只有两组链接放在nav里，另外两组则由于不是主要的导航而没有放在 nav 里

3.7 标记页面的主要区域

大多数网页都有一些不同的区块，如页眉、页脚、包含额外信息的附注栏、指向其他网站的链接，等等。不过，一个页面只有一个部分代表其主要内容。可以将这样的内容包在 main 元素中，该元素在一个页面仅使用一次（参见图 3.7.1 和图 3.7.2）。

3

```
...
<body>
<header role="banner">
    <nav role="navigation">
        ... [包含多个链接的ul] ...
    </nav>
</header>

<main role="main">
    <article>
        <h1 id="gaudi">Barcelona's Architect
        ➝ </h1>
        <p>Antoni Gaudí's incredible
        ➝ buildings bring millions ...</p>

        ... [页面主要区域的其他内容] ...
    </article>
</main>

<aside role="complementary">
    <h1>Architectural Wonders of Barcelona
    ➝ </h1>

    ... [附注栏的其他内容] ...
</aside>

<footer role="contentinfo">
    ... [版权] ...
</footer>
</body>
</html>
```

图 3.7.1 这是本章反复使用的页面的完整代码的主体结构。你可能还不熟悉其中某些元素，不过随后就会讲到。main 元素包围着代表页面主题的内容。最好在 main 开始标签中加上 role="main"（参见最后一条提示）

提示 main 元素是 HTML5 新添加的元素。记住，在一个页面里仅使用一次。



图 3.7.2　同 p、header、footer 等元素一样，main 元素的内容显示在新的一行，除此之外不会影响页面的任何样式。（这里显示的布局并不是由 main 产生的，而是由 CSS 控制的。）这里为了方便你在应用了样式的完整页面中识别出 main，特意为其添加了蓝色的边框

提 示　如果创建的是 Web 应用，则使用 main 包围其主要的功能。

提 示　不能将 main 放置在 article、aside、footer、header 或 nav 元素中。

提 示　如图 3.7.1 所示，role="main" 可以帮助屏幕阅读器定位页面的主要区域。要进一步了解这样的 ARIA 地标，参见 3.13 节。

3.8　创建文章

HTML5 的另一个新元素便是 article，如图 3.8.1 和图 3.8.2 所示。你已经见过一些用到该元素的示例。现在，让我们深入了解一下它的作用。

```
...
<body>
<header role="banner">
    <nav role="navigation">
       ... [包含多个链接的ul] ...
    </nav>
</header>

<main role="main">
    <article>
       <h1 id="gaudi">Barcelona's Architect
       → </h1>

       <p>Antoni Gaudí's incredible buildings
       → bring millions of tourists to
       → Barcelona each year.</p>

       <p>Gaudí's non-conformity, already
       → visible in his teenage years,
       → coupled with his quiet but firm
       → devotion to the church, made a
       → unique foundation for his thoughts
       → and ideas. His search for simplicity
       → ...is quite apparent in his work,
       → from the <a href="#park-guell">Park
       → Guell</a> and its incredible
       → sculptures and mosaics, to...</p>

       <h2 id="sagrada-familia" lang="es">
       → La Sagrada Família</h2>

       <p><img src="img/towers.jpg"
       → width="75" height="100" alt="Sagrada
       → Família Towers" /> The
       → complicatedly named and curiously
       → unfinished masterpiece...</p>

       <h2 id="park-guell">Park Guell</h2>

       ... [图像和段落] ...
    </article>
</main>
</body>
</html>
```

图 3.8.1　为了精简，我对文章内容进行了缩写，略去了与上一节相同的 nav 代码。可以在本书网站查看完整的代码（www.htmlcssvqs.com/8ed/structure-final）。尽管在这个例子里只有段落和图像，但 article 可以包含各种类型的内容



图 3.8.2　现在，页面有了 header、nav、main 和 article 元素，以及它们各自的内容。在不同的浏览器中，article 中标题的字号可能不同。可以应用 CSS 使它们在不同的浏览器中显示相同的大小（参见第 10 章）

根据其称，你大概会猜想 article 用于包含像报纸文章一样的内容。不过，article 并不局限于此。在 HTML5 中，“文章”（article）并非这个词本来的意思。

HTML5 对该元素的定义如下：

> article 元素表示文档、页面、应用或网站中一个独立的容器，原则上是可独立分配或可再用的，就像聚合内容中的各部分。它可以是一篇论坛帖子、一篇杂志或报纸文章、一篇博客条目、一则用户提交的评论、一个交互式的小部件或小工具，或者任何其他独立的内容项。

其他 article 的例子包括电影或音乐评论、案例研究、产品描述，等等。你或许惊讶于它还可以是交互式的小部件或小工具，不过这些确实是独立的、可再分配的内容项。

创建文章的步骤

(1) 输入 <article>。

(2) 输入文章的内容。内容可以包含任意数量的元素。元素类型包括段落、列表、音频、视频、图像、图形等。

(3) 输入 </article>。

提示　可以将 article 嵌套在另一个 article 中，只要里面的 article 与外面的 article 是部分与整体的关系。下文提供了这样的一个例子。

提示　一个页面可以有多个 article 元素（也可以没有）。例如，博客的主页通常包括几篇最新的文章，其中每一篇都是其自身的 article。

提示　一个 article 可以包含一个或多个 section 元素（下一节会介绍该元素）。在 article 里包含独立的 h1 ～ h6 也是很好的做法，就像示例代码那样。

更多 article 示例

图 3.8.1 中的示例只是使用 article 的一种方式，下面看看其他的用法。

图 3.8.3 展示了对基本的新闻报道或报告进行标记的方法。注意 footer 和 address 元素的使用（对它们的讨论分别参见本章和第 4 章）。这里，address 只应用于其父元素 article（即这里显示的 article），而非整个页面或任何嵌套在那个 article 里面的 article（如图 3.8.4 中的读者评论）。

图 3.8.4 展示了嵌套在父元素 article 里面的 article 元素。该例中嵌套的 article 是用户提交的评论，就像你在博客或新闻网站上见到的评论部分。该例还显示了 section 元素（参见 3.9 节）和 time 元素（在第 4 章讨论）的用法。

这些只是使用 article 及有关元素的几个常见方式。

```
...
<article>
    <h1>The Diversity of Papua New Guinea
    ➝ </h1>
    <p>Papua New Guinea is home to more than
    ➝ 800 tribes and languages ...</p>

    ... [其余内容] ...

    <!-- 文章的页脚，并非页面级的页脚 -->
    <footer>
       <p>Leandra Allen is a freelance
       ➝ journalist who earned her degree
       ➝ in anthropology from the University
       ➝ of Copenhagen.</p>

       <address>
       You may reach her at <a href="mailto:
       ➝ leandra@therunningwriter.com">
       ➝ leandra@therunningwriter.com</a>.
       </address>
    </footer>
</article>
...
```

图 3.8.3　标记包含作者信息的文章的常见方式

```
...
<article>
    <h1>The Diversity of Papua New Guinea
    ➝ </h1>
    ... [父元素article内容] ...

    <footer>
       ... [父元素article页脚] ...
    </footer>

    <section>
       <h2>Reader Comments</h2>
       <article>
          <footer>travelgal wrote on
          ➝ <time datetime="2014-02-26">
          ➝ February 26, 2014</time>:
          ➝ </footer>
          <p>Great article! I've always
          ➝ been curious about Papua New
          ➝ Guinea.</p>
       </article>

       <article>
          ... [下一则读者评论] ...
       </article>
    </section>
</article>
...
```

图 3.8.4　每条读者评论都包含在一个 article 里，这些 article 元素则嵌套在主 article 里

3.9　定义区块

另一个 HTML5 的新元素是 section，参见图 3.9.1 至图 3.9.3。HTML5 对该元素的部分定义如下：

> section 元素代表文档或应用的一个一般的区块。在这里，section 是具有相似主题的一组内容，通常包含一个标题。
>
> section 的例子包含章节、标签式对话框中的各种标签页、论文中带编号的区块。比如网站的主页可以分成介绍、新闻条目、联系信息等区块。

尽管我们将 section 定义成“通用的”区块，但不要将它与 div 元素（参见 3.12 节）混淆。从语义上讲，section 标记的是页面中的特定区域，而 div 则不传达任何语义。

```
...
<body>
...
<main role="main">
    <h1>Latest World News</h1>
    <section>
        <h2>Breaking News</h2>
        <ul>... [标题列表] ...</ul>
    </section>

    <section>
        <h2>Business</h2>
        <ul>... [标题列表] ...</ul>
    </section>

    <section>
        <h2>Arts</h2>
        <ul>... [标题列表] ...</ul>
    </section>
</main>
...
</body>
</html>
```

图 3.9.1　几乎任何新闻网站都会对新闻进行分类。每个类别都可以标记为一个 section

```
...
<h1>Graduation Program</h1>
<section>
    <h2>Ceremony</h2>
    <ol>
        <li>Opening Procession</li>
        <li>Speech by Valedictorian</li>
        <li>Speech by Class President</li>
        ...
    </ol>
</section>

<section>
    <h2>Graduates (alphabetical)</h2>
    <ol>
        <li>Molly Carpenter</li>
        ...
    </ol>
</section>
...
```

图 3.9.2　这个例子是在 HTML5 规范中一个例子的基础上修改而成的，其中，section 元素用于标记一次毕业活动安排中的不同部分

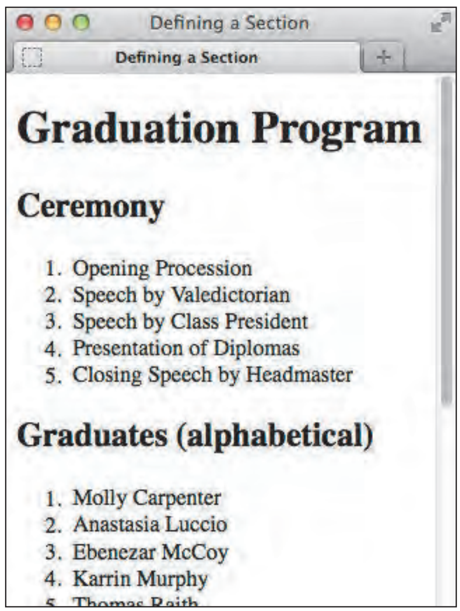


图 3.9.3　跟本章很多其他的元素一样，section 并不影响页面的显示。这里用数字编号展示各个条目是因为图 3.9.2 中使用了有序列表（ol）。关于列表，参见第 15 章

定义区块的步骤

(1) 输入 <section>。

(2) 输入区块的内容。内容可以包含任意数量的元素。元素类型包括段落、列表、音频、视频、图像、图形等。

(3) 输入 </section>。

提示　如果只是出于添加样式的原因要对内容添加一个容器，应使用 div 而不是 section。

提示　可以将 section 嵌套在 article 里，从而显式地标出报告、故事、手册等文章的不同部分或不同章节。例如，可以在本章所示 Antoni Gaudí 的例子中使用 section 元素，即使用一个 section 包围 La Sagrada Família 这个 h2 及相关的段落，用另一个 section 包围 Park Guell 这个 h2 及相关的段落。

体会 section 元素

我特意引用了 HTML5 对 section 的定义，以便你对该元素有个最为直观的理解。section 元素可能是 Web 社区讨论最激烈的主题，因为似乎很难解释清楚它的用法。

在考虑何时使用 section 的时候，记住定义中“具有相似主题的一组内容”这一条是很有帮助的。这也是 section 区别于 div 的另一个原因。section 和 article 的区别在于，section 在本质上组织性和结构性更强，而 article 代表的是自包含的容器。

正如在第 1 章中提到的，对内容进行标记时，并非总能分出对和错，只是大多数时候有正确的选择。而其他时候，就只能依靠个人对最适合描述内容的 HTML 元素的感觉了。

在考虑是否使用 section 的时候，一定要仔细思考，不过也不必每次都对是否用对感到担心。有时，些许主观并不会影响页面正常工作。而且，也不会有人半夜来敲你的门。

哦，我可能会，不过这只是因为外面很黑，很吓人。

3.10　指定附注栏

有时候，页面中有一部分内容与主体内容相关性没有那么强，但可以独立存在（参见图 3.10.1 和图 3.10.2），如何在语义上表示出来呢?

在 HTML5 之前一直无法显式地做到这一点。在 HTML5 中，我们有了 aside 元素。

使用 aside 的例子还包括重要引述、侧栏（图 3.10.3）、指向相关文章的一组链接（通常针对新闻网站）、广告、nav 元素组（如博客的友情链接），Twitter 源、相关产品列表（通常针对电子商务网站），等等。

尽管我们很容易将 aside 元素看做侧栏，

但该元素其实可以用在页面的很多地方，具体依上下文而定。如果 aside 嵌套在页面主要内容内（而不是作为侧栏位于主要内容之外），则其中的内容应与其所在的内容密切相关，而不是仅与页面整体内容相关。“其他 aside 的例子”中的例 1 就是这样的。

```
...
<body>
<header role="banner">
     <nav role="navigation">
        ... [包含多个链接的ul] ...
     </nav>
</header>

<main role="main">
     <article>
        <h1 id="gaudi">Barcelona's Architect</h1>

     ... [文章的其他部分] ...
     </article>
</main>

<aside role="complementary">
     <h1>Architectural Wonders of Barcelona</h1>

     <p>Barcelona is home to many architectural wonders in addition to Gaudí's work. Some of them include:</p>
     <ul>
        <li lang="es">Arc de Triomf</li>
        <li>The cathedral <span lang="es">(La Seu)</span></li>
        <li lang="es">Gran Teatre del Liceu</li>
        <li lang="es">Pavilion Mies van der Rohe</li>
        <li lang="es">Santa Maria del Mar</li>
     </ul>

     <p><small>Credit: <a href="http://www.barcelona.de/en/barcelona-architecture-buildings.html"
     ➝ rel="external"><cite>Barcelona.de</cite></a>.</small></p>
</aside>
</body>
</html>
```

图 3.10.1 这个 aside 是有关巴塞罗那建筑奇迹的信息，与页面主要关注的 Antoni Gaudí 内容相关性稍差，且可以在没有这个上下文的情况下独立存在。我可以将它嵌套在 article 里面，因为它们是相关的，但最终我将它放在了 article 后面，使用 CSS 让它看起来像侧栏（图 3.10.3）。aside 里面的 role="complementary" 是可选的，但它可以提高可访问性。要了解更多信息，参见最后一条提示



图 3.10.2　aside 出现在文章的下面，因为在 HTML 中 aside 就跟在 article 的后面，参见图 3.10.1。可以看到，浏览器在默认情况下并未对 aside 应用任何特殊样式（除了从新行开始）。不过，可以通过 CSS 控制其外观显示（图 3.10.3）

指定附注栏的步骤

(1) 输入 <aside>。

(2) 输入附注栏的内容。内容可以包含任意数量的元素。元素类型包括段落、列表、音频、视频、图像、图形等。



图 3.10.3　对已完成的页面应用 CSS 时，可以让 aside（以 Architectural Wonders of Barcelona 开头）显示在主要内容的旁边，而不是下边。这个例子已将 aside 做成了侧栏（我们将在第 11 章学习如何制作两栏的布局）

(3) 输入 </aside>。

提 示　在 HTML 中，应该将附注栏内容放在 main 的内容之后（图 3.10.1）。出于 SEO 和可访问性的目的，最好将重要的内容放在前面。可以通过 CSS 改变它们在浏览器中的显示顺序。

提 示　对于与内容有关的图像（如图表、图形或带有说明文字的插图），使用 figure（参见第 4 章）而非 aside。

提 示　HTML5 不允许将 aside 嵌套在 address 元素内。

提 示　要了解如何在 aside 中使用 role="complementary"，参见 3.13 节。

其他 aside 的例子

上面提到，aside 可以嵌套在主要内容里面，也可以位于主要内容外面。

例 1（嵌套在主要内容中）

```
...
<body>
<main role="main">
    <article>
        <h1>The Diversity of Papua New Guinea</h1>
        ... [ 文章内容 ] ...
        <aside>
            <h2>Papua New Guinea Quick Facts</h2>
            <ul>
              <li>The country has 38 of the 43 known birds of paradise</li>
              <li>Though quite tropical in some regions, others occasionally
              ➝ experience snowfall.</li>
              ...
            </ul>
        </aside>
    ... [ 更多文章内容 ] ...
    </article>
</main>
</body>
</html>
```

如果这个 article 要包含一个重要引述，也应该包在 aside 里。也可以弄一个“相关故事”的 aside，包含一组指向关于该国家的其他文章的链接。另外，aside 还可以成为页面的另一个区块，而不是嵌套在 main 中的 article 里。

我们已看到了 aside 位于附注栏的例子（图 3.10.1 和图 3.10.3）。现在，考虑一套设计作品或一组案例研究，其中每个 HTML 页面展示一个项目，并在毗邻的一栏中（显示位置由 CSS 控制，而不是由例 2 中的代码控制）提供指向其他项目页面的链接（嵌套在 nav 中）。

例 2（aside 并未嵌套在主要内容中，且包含一个 nav）

```
...
<body>
<main role="main">
    <article>
        <h1>... [ 项目名称 ] ...</h1>
        <figure>... [ 项目图片 ] ...</figure>
        <p>... [ 项目报道 ] ...</p>
    </article>
</main>
```

```
<!-- 这个附注栏没有嵌套在主要内容中 -->
<aside>
    <h2>Other Projects</h2>
    <nav>
        <ul>
            <li><a href="habitat-for-humanity.html">Habitat for Humanity
            ➝ brochure</a></li>
            <li><a href="royal-philharmonic.html">Royal Philharmonic Orchestra
            ➝ website</a></li>
            ...
        </ul>
    </nav>
</aside>
</body>
</html>
```

这个 aside 位于 article 的外面，因为它与页面整体内容相关，但与 article 的内容相关性没那么强。

3.11 创建页脚

当你想到页脚的时候，你大概想的是页面底部的页脚（通常包括版权声明，可能还包括指向隐私政策页面的链接以及其他类似的内容）。HTML5 的 footer 元素可以用在这样的地方，但它同 header 一样，还可以用在其他的地方。

footer 元素代表嵌套它的最近的 article、aside、blockquote、body、details、fieldset、figure、nav、section 或 td 元素的页脚。只有当它最近的祖先是 body 时，它才是**整个**页面的页脚（参见图 3.11.1 和图 3.11.2）。

如果一个 footer 包着它所在区块（如一个 article）的**所有**内容，它代表的是像附录、索引、版权页、许可协议这样的内容。

创建页脚的步骤

(1) 将光标放在希望创建页脚的元素里。

(2) 输入 <footer>。

(3) 输入页脚的内容。

(4) 输入 </footer>。

提示 页脚通常包含关于它所在区块的信息，如指向相关文档的链接、版权信息、作者及其他类似条目。参见“其他 footer 示例”中的前两个例子。

提示 页脚并不一定要位于所在元素的末尾，不过通常是这样的。

```
...
<body>
<header role="banner">
     <nav role="navigation">
        ... [包含链接的无序列表] ...
     </nav>
</header>

<main role="main">
     <article>
        <h1 id="gaudi">Barcelona's Architect
        ➝ </h1>
        ... [文章的其他部分] ...

        <h2 id="sagrada-familia" lang="es">
        ➝ La Sagrada Família</h2>
         ... [图像和段落] ...

        <h2 id="park-guell">Park Guell</h2>
        ... [其他图像和段落] ...
     </article>
</main>

<aside role="complementary">
     <h1>Architectural Wonders of Barcelona
     ➝ </h1>
     ... [附注栏的其他部分] ...
</aside>

<footer>
     <p><small>&copy; Copyright All About
     ➝ Gaudí</small></p>
</footer>
</body>
</html>
```

图 3.11.1 这个 footer 代表整个页面的页脚，因为它最近的祖先是 body 元素。现在，页面有了 header、nav、main、article、aside 和 footer 元素。并非每个页面都需要以上所有元素，但它们确实代表了 HTML 中的主要页面构成要素

提示 不能在 footer 里嵌套 header 或另一个 footer。同时，也不能将 footer 嵌套在 header 或 address 元素里。

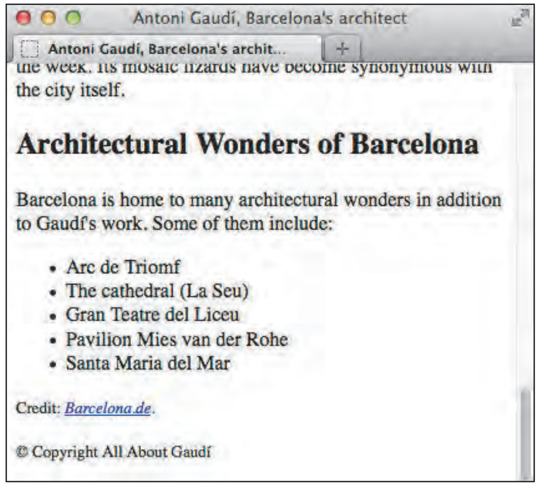


图 3.11.2 这个 footer 出现在页面的最底部，位于 aside 之后。footer 元素本身不会为文本添加任何默认样式。这里，版权信息的字号比普通文本的小，这是因为它嵌套在“用来对法律文本进行语义化”的 small 元素里（参见第 4 章）。像其他内容一样，可以通过 CSS 修改 footer 元素所含内容的字号

提示 要了解 footer 如何取代 HTML5 之前版本中 div 元素的一个功能，参见 3.12 节。

提示 要了解在哪种情况下可对页面级的 footer 使用 role="contentinfo"，参见 3.13 节。将其放入图 3.11.1 中的代码是合理的，因为它代表的是整个页面的页脚，不过我有意略去了，避免给读者留下所有 footer 元素都需要 role="contentinfo" 的印象。在“其他 footer 示例”中，我们在不同的示例中体现了这种区别，另外还演示了使用 role 属性的正确方法。

其他 footer 示例

我们已经学习了页面级 footer 的一个小例子（图 3.11.1 和图 3.11.2）。图 3.11.3 是另一个页面级 footer 的例子，不过其中的内容更多一些。

图 3.11.4 展示了位于页面区块 article 中的一个 footer，以及作为整个页面页脚的另一个 footer。（对 address 元素的解释，参见“更多 article 示例”。）

注意只有页面级页脚有 role="contentinfo"。参见 3.13 节了解此功能。

```
...
<body>
... [包含全局导航的页面级页眉] ...
... [页面内容] ...

<!-- 这是一个页面级的页脚，因为它最近的祖先元素
     是body -->
<footer role="contentinfo">
     <p><small>&copy; Copyright 2014 The
     ➝ Corporation, Inc.</small></p>

     <ul>
        <li><a href="terms-of-use.html">
        ➝ Terms of Use</a></li>
        <li><a href="privacy-policy.html">
        ➝ Privacy Policy</a></li>
     </ul>
</footer>
</body>
</html>
```

图 3.11.3 页面级的 footer 通常包含版权信息以及不属于页面全局导航（位于 header 里）的一些链接

```
...
<body>
...
<article>
     <h1>... [文章标题] ...</h1>
     <p>... [文章内容] ...</p>

     <!-- 文章的页脚 -->
     <footer>
        <p>Leandra Allen is a freelance
        ➝ journalist who earned her
        ➝ degree in anthropology from the
        ➝ University of Copenhagen.</p>
        <address>
           You may reach her at
           ➝ <a href="mailto:leandra@
           ➝ therunningwriter.com">
           ➝ leandra@therunningwriter.com
           ➝ </a>.
        </address>
     </footer>
</article>

<!-- 页面级页脚 -->
<footer role="contentinfo">
     ... [版权信息等] ...
</footer>
</body>
</html>
```

图 3.11.4 第一个 footer 包含在 article 内，因此是属于该 article 的页脚。第二个 footer 是页面级的。只能对页面级的 footer 使用 role="contentinfo"，且一个页面只能使用一次

3.12　创建通用容器

有时需要在一段内容外围包一个容器，从而可以为其应用 CSS 样式或 JavaScript 效果。如果没有这个容器，页面就会不一样（参见图 3.12.1）。在评估内容的时候，考虑使用 article、section、aside、nav 等元素，却发现它们从语义上讲都不合适。

这时，你真正需要的是一个通用容器，一个完全没有任何语义含义的容器。这个容器就是 div（来自 division 一词）元素，如图 3.12.2 所示。有了 div，就可以为其添加样式（参见图 3.12.3）或 JavaScript 效果了。一定要阅读补充材料“有关 div 的一些历史以及何时在 HTML5 中使用它”，了解什么时候可以在页面中使用 div。



图 3.12.1　在不使用 div 元素的情况下，我们实现了这个设计。不过，为页面内容加上 div（参见图 3.12.2）以后，我们有了一个可以添加更多样式的通用容器（效果如图 3.12.3 所示）

```
...
<body>
<div>
    <header role="banner">
        <nav role="navigation">
            ... [包含多个链接的无序列表] ...
        </nav>
    </header>

    <main role="main">
        <article>
            <h1 id="gaudi">Barcelona's
            ➝ Architect</h1>
            ... [文章的其余内容] ...

            <h2 id="sagrada-familia" lang="es">
            ➝ La Sagrada Família</h2>
            ... [图像和段落] ...

            <h2 id="park-guell">Park Guell</h2>
            ... [另一个图像和段落] ...
        </article>
    </main>

    <aside role="complementary">
        <h1>Architectural Wonders of
        ➝ Barcelona</h1>
        ... [aside的其余内容] ...
    </aside>

    <footer role="contentinfo">
        <p><small>&copy; Copyright All About
        ➝ Gaudí</small></p>
    </footer>
</div>
</body>
</html>
```

图 3.12.2　现在有一个 div 包着所有的内容。页面的语义没有发生改变，但现在我们有了一个可以用 CSS 添加样式的通用容器（图 3.12.3）



图 3.12.3　div 元素自身没有任何默认样式，只是其包含的内容从新的一行开始（图 3.12.4）。不过，我们可以对 div 添加样式以实现自己的设计。这里，我对 div 添加了浅的背景色和阴影，这样我就可以将 body 元素的背景改为渐变的红色，使内容凸显出来。要查看实现这个页面的 HTML 和 CSS，可以访问 www.htmlcssvqs.com/8ed/structure-final

创建通用容器的步骤

(1) 输入 <div>。

(2) 创建容器的内容，内容可以包含任意数量的元素。

(3) 在容器的结尾处输入 </div>。

提 示　同 header、footer、main、article、section、aside、nav、h1 ~ h6、p 和其他很多元素一样，默认情况下，div 从新的一行开始显示（图 3.12.4）。

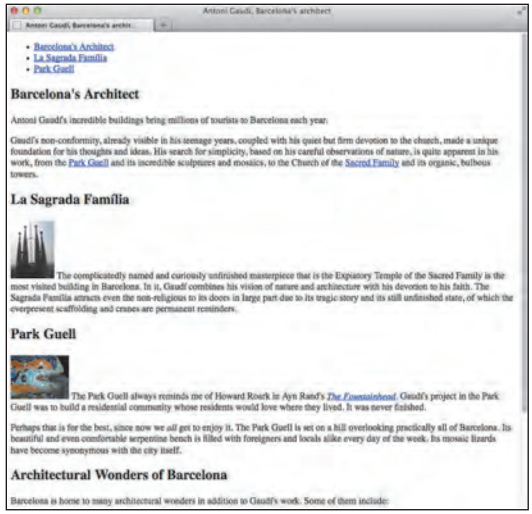


图 3.12.4　同一个页面在未对 div、标题、段落及其他元素添加 CSS 的样子。可以看到，div 并未产生任何独特的样式

提 示　div 对使用 JavaScript 实现一些特定的交互行为或效果也是有帮助的。例如，在页面中展示一张照片或一个对话框，同时让背景页面覆盖一个半透明的层（这个层通常是一个 div）。

提 示　尽管我始终强调 HTML 用于对内容的含义进行描述，但 div 并不是唯一没有语义价值的元素。span 是与 div 对应的一个元素：div 是块级内容的无语义容器，而 span（写作 这里是内容 ）则是短语内容的无语义容器，例如它可以放在段落元素 p 之内。参见第 4 章了解 span。

提 示　要了解如何在 div 中使用地标角色，参见 3.13 节。

有关 div 的一些历史以及何时在 HTML5 中使用它

在本章讲到的结构性元素中，div 是除了 h1 ~ h6 以外唯一早于 HTML5 出现的元素。在 HTML5 之前，div 是包围大块内容（如页眉、页脚、主要内容、插图、附注栏等）从而可用 CSS 为之添加样式的不二选择。之前 div 没有任何语义含义，现在也一样。

这就是 HTML5 引入 header、footer、main、article、section、aside 和 nav 的原因。这些类型的构造块在网页中普遍存在，因此它们可以成为具有**独立含义**的元素。在 HTML5 中，div 并没有消失，只是使用它的场合变少了。

让我们看几个使用 div 的常见示例。

我们之前已经见过一个了：为添加样式，用一个容器将整个页面包起来（参见图 3.12.2 和图 3.12.3）。

如何用 div 实现两栏布局呢？我对 article 和 aside 元素分别添加一些 CSS，就能让它们各自成为一栏（我们将从第 7 章开始学习 CSS；关于用 CSS 进行布局，参见第 11 章）。

然而，大多数情况下，每一栏都有不止一个区块的内容。例如，主要内容区第一个 article 下面可能还有另一个 article（或 section、aside 等）。又如，也许你想在第二栏再放一个 aside 显示指向关于 Gaudí 的其他网站的链接，或许再加一个其他类型的元素。

你需要将期望出现在同一栏的内容包在一个 div 里（图 3.12.5），然后对这个 div 添加相应的样式。（你或许在想是否也可以用 section，但要知道该元素并不能作为添加样式的通用容器。）图 3.12.6 有助于你形象地理解代码和期望呈现的 CSS 布局之间的关系。记住这只是这段 HTML 可能形成的布局之一。要知道 CSS 是相当强大的。

因此，将希望在样式上组成一栏的内容用 div 包起来是一种非常常见的做法（当然，也可以用 div 包住两个以上的栏）。至于放在其中的内容，则可能差别极大，视页面内容安排而定。别忘了，作为内容区块的主要语义化容器，article、section、aside 和 nav 可以位于任何地方。此外，header 和 footer 也是这样。在图 3.12.5 和图 3.12.6 中，main 中只有 article 元素，附注栏只有 aside 元素，但不要对此作过分解读。

不过，可以肯定的是，div 应该作为最后一个备用容器，因为它没有任何语义价值。大多数时候，使用 header、footer、main（仅使用一次）、article、section、aside 甚至 nav 代替 div 会更合适。但是，如果语义上不合适，也**不必**为了刻意避免使用 div 而使用上述元素。有用得上 div 的地方，只是需要限制其使用。

说到这里，有一种情况，div 适合所有（或几乎所有，这由你决定）页面容器，而无需使用新的 HTML5 元素。更多信息参见 11.3 节。

3

邮电

```
...
<body>
<!-- 开始页面容器 -->
<div class="container">
    <header role="banner">
        ...
    </header>

    <!-- 应用CSS后的第一栏 -->
    <main role="main">
        <article>
            ...
        </article>

        <article>
            ...
        </article>

        ... [更多的区块] ...
    </main>
    <!-- 结束第一栏 -->

    <!-- 应用CSS后的第二栏 -->
    <div class="sidebar">
        <aside role="complementary">
            ...
        </aside>

        <aside role="complementary">
            ...
        </aside>

        ... [更多的区块] ...
    </div>
    <!-- 结束第二栏  -->

    <footer role="contentinfo">
        ...
    </footer>
</div>
<!-- 结束页面容器 -->
</body>
</html>
```

图 3.12.5　这个页面有一个包含整个页面的 div，另外还新增了一个包着附注栏的 div。带有 class="sidebar"（使用任何类名都可以）的 div 包围着想要显示为第二栏的内容。然后可以在 CSS 中通过类定位到这样的 div 并对其应用样式。main 则加上了显示为第一栏的样式

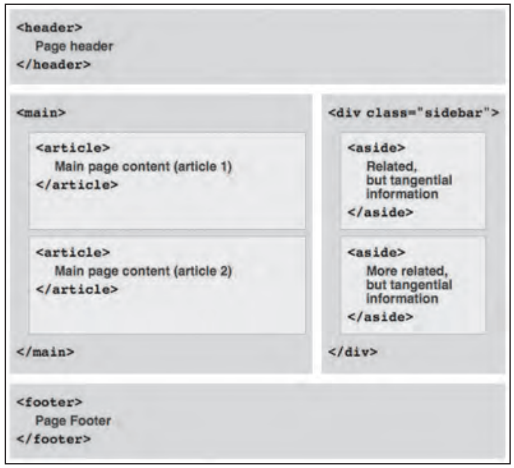


图 3.12.6　此图显示了图 3.12.5 中的代码如何在概念上对应到 CSS 布局对应的形式。这是很常见的一种安排，但只是针对这段 HTML 众多可能的 CSS 样式中的一种。一定要阅读 3.13 节，了解如何进一步增强页面的语义和可访问性

3.13　使用 ARIA 改善可访问性

WAI-ARIA（Web Accessibility Initiative's Accessible Rich Internet Applications，无障碍网页倡议 – 无障碍的富互联网应用，也简称 ARIA）是一种技术规范，自称“有桥梁作用的技术”。之所以这样说，是因为在 HTML 提供相应的语义功能之前，它会使用属性来填补一些语义上的空白。

你可能已经注意到，本章的大多数例子都使用了一个或多个 ARIA role 属性，而且多次提示你跳到这一节了解更多关于该属性的说明。现在，就让我们开始吧！噢，等等，在讲解 role 属性之前，先简单看一下无障碍访问的知识。

无障碍访问的意义是让所有的访问者都能获取网站的内容。你的网站的一部分访问者可能需要借助辅助设备（如屏幕阅读器）访问你的页面内容（参见补充材料“试用屏

幕阅读器”）。

让网站具备无障碍访问功能是成为严肃负责的 Web 公民的重要方面。而且，这样做对你也是有好处的——为什么**不让**访问者能够获取网站的内容呢？

幸运的是，在大多数情况下，让页面具有无障碍访问功能并不难。只要对内容使用最恰当的 HTML 进行标记，就能改进网站的可访问性。如果你一直在这样做，非常好。在这一节，我会讲到如何在 HTML 中添加一些简单的属性进一步提升网站的可访问性。

地标角色

前面提到，ARIA 填补了 HTML 的语义空白。例如，如果你想让屏幕阅读器用户知道如何跳至页面级的页脚，应该使用什么 HTML 标记呢？标记为 `footer` 还不够，要知道页面可以包含多个 `footer`。

对此，HTML 没有解决方案，而 ARIA 的**地标角色**（landmark role）则可以满足需求。地标角色可以帮助用户识别页面区域，从而让屏幕阅读器用户可以直接跳到这些区域。通常，对这些区域指定 `role` 属性就可以了。

如表 3.13.1 所示，地标角色和 HTML5 元素之间有一些重合（要知道 HTML5 也使用本章介绍的一些新元素填补了某些语义空白），但目前屏幕阅读器对 ARIA 的支持更多。因此你可以继续按既有的方式创建 HTML（包括使用新元素），同时添加一些 ARIA 角色提升页面的可访问性。

试用屏幕阅读器

以下屏幕阅读器是最为常见的一些。除了 VoiceOver 以外，这些屏幕阅读器都仅适用于 Windows。

- JAWS：免费的演示版位于 www.freedomscientific.com。
- NVDA：免费，位于 www.nvda-project.org。
- VoiceOver：免费，是 OS X 和 iOS 4+ 的一部分。在 OS X 上，可以通过 Command-F5 开启或关闭 VoiceOver。要了解如何在 iOS 设备上使用 VoiceOver，请访问 https://support.apple.com/kb/HT3598。
- Window-Eyes：免费的演示版位于 www.gwmicro.com。

强烈建议你从这些软件中挑选一个进行试用。这样，你会对屏幕阅读器的用户体验有更深的理解。而且你还将直观地了解到语义化的 HTML 如何影响使用屏幕阅读器的体验。在屏幕阅读器中测试网页最好成为建站标准流程的一部分。

表 3.13.1　一些可用的地标角色

地标角色	如何使用及何时使用
`role="banner"`（横幅） 面向全站的内容，通常包含网站标志、网站赞助者标志、全站搜索工具等。横幅通常显示在页面的顶端，而且通常横跨整个页面的宽度	将其添加到页面级的 `header` 元素，每个页面只用一次
`role="navigation"`（导航） 文档内不同部分或相关文档的导航性元素（通常为链接）的集合	与 `nav` 元素是对应关系。应将其添加到每个 `nav` 元素，或其他包含导航性链接的容器。这个角色可在每个页面上使用多次，但是同 `nav` 一样，不要过度使用该属性

（续）

地标角色	如何使用及何时使用
role="main"（主体） 文档的主要内容	与 main 元素是对应关系。最好将其添加到 main 元素，也可以添加到其他表示主体内容的元素（可能是 div）。在每个页面仅使用一次
role="complementary"（补充性内容） 文档中作为主体内容补充的支撑部分。它对区分主体内容是有意义的	与 aside 元素是对应关系。应将其添加到 aside 或 div 元素（前提 是该 div 仅包含补充性内容）。可以在一个页面里包含多个 complementary 角色，但不要过度使用
role="contentinfo"（内容信息） 包含关于文档的信息的大块可感知区域这类信息的例子包括版权声明和指向隐私权声明的链接等	将其添加至整个页面的页脚（通常为 footer 元素）。每个页面仅使用一次

使用地标角色很简单。图 3.13.1 与 3.12 节中的例子几乎是一样的，只不过添加了一个 nav 元素，并对地标角色进行了突出显示。

表 3.13.1 引用了一些 ARIA 规范（www.w3.org/TR/wai-aria/roles#landmark_roles） 中对地标角色的定义及推荐用法（另外还有三种地标角色，参见第一条提示）。对它们的描述看起来会很熟悉，因为它们与对应的 HTML 元素的用法是相似的（参见图 3.13.1 和图 3.13.2）。

```
...
<body>
<!-- 开始页面容器 -->
<div class="container">
     <header role="banner">
        ...
        <nav role="navigation">
           ... [包含多个链接的无序列表] ...
        </nav>
     </header>

     <!-- 应用CSS后的第一栏 -->
     <main role="main">
        <article>
           ...
        </article>

        <article>
           ...
        </article>

        ... [其他区块] ...
     </main>
```

```
     <!-- 结束第一栏 -->

     <!-- 应用CSS后的第二栏 -->
     <div class="sidebar">
        <aside role="complementary">
           ...
        </aside>

        <aside role="complementary">
           ...
        </aside>

        ... [其他区块] ...
     </div>
     <!-- 结束第二栏 -->

     <footer role="contentinfo">
        ...
     </footer>
</div>
<!-- 结束页面容器 -->
</body>
</html>
```

图 3.13.1　这个页面使用了表 3.13.1 中列出的五种地标角色（还有一种表中未列出的）

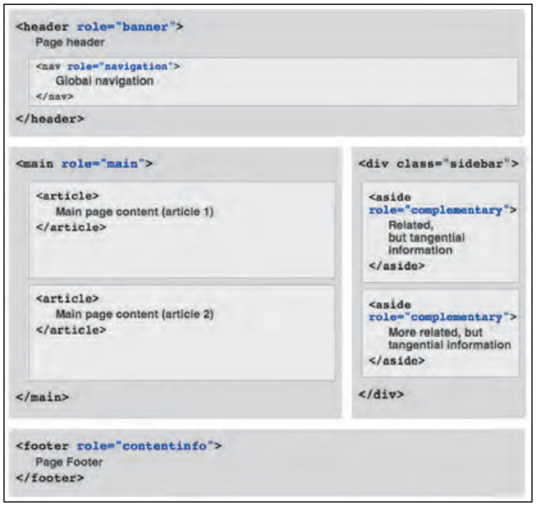


图 3.13.2　这张图与 3.12 节中的图很相似。这张图仅用于将常见的地标角色可视化，并不代表它们对布局有影响。例如，在 main 元素中，可以包含一个带有 role="navigation" 的 nav 作为局部导航。同时，带有 class="sidebar" 的 div 也可以添加 role="complementary"，从而代替 aside 元素，前提是整个附注栏的内容确实为补充性内容

需要澄清的是，即便不使用 ARIA 地标角色，页面看起来也没有任何差别，但是使用它们可以提升使用辅助设备的用户的体验。出于这个理由，推荐使用它们。在本书的例子中，我都在恰当的地方使用了它们。本书配套网站上的代码示例也是这样做的。

提示　表 3.13.1 没有包含另外三种地标角色。对表单元素来说，form 角色是多余的；search 用于标记搜索表单；application 则属于高级用法。16.8 节有一个使用 role="search" 的例子。

提示　不要在页面上过多地使用地标角色。过多的地标角色会让屏幕阅读器用户感到累赘，从而降低地标的作用，影响整体体验。

提示　可用性专家 Steve Faulkner 对地标角色有详细的讲解，见 blog.paciellogroup.com/2013/02/using-wai-aria-landmarks-2013/。（表 3.13.1 与他的表格之间的细小差别可以忽略。）他还引用了另一位专家 Léonie Watson 录制的演示屏幕阅读器用户访问页面行为的视频，推荐观看！

提示　WebAIM 会定期对屏幕阅读器用户做一些调查，以了解他们访问网站的偏好和遇到的问题。调查结果很值得一读，见 webaim.org/projects/screenreadersurvey4/。

提示　地标角色只是 ARIA 规范（www.w3.org/TR/wai-aria/）众多特性中的一个。如果对相关的实施指南感兴趣，可以访问 www.w3.org/WAI/PF/aria-practices/。

提示　可以在 CSS 选择器中使用 ARIA 角色属性，从而对使用它们的元素添加样式。具体细节参见第 11 章。

3.14　为元素指定类别或 ID 名称

尽管并非必需，但是可以给 HTML 元素分配唯一的标识符（ID），或指定其属于某个（或某几个）类别，也可以同时指定标识符和类别，如图 3.14.1 所示。

接着，就可以对具有给定 id 或 class 名称的元素添加样式了（但一般不推荐出于添加样式的目的使用 id）；或者创建指向具有特定 id 的元素的链接（图 3.14.1）；还可以使用 JavaScript 获取 id 和 class 属性，从而对元素添加特定的行为。

```
...
<body>
<div class="container">
    <header role="banner">
       <nav role="navigation">
          <ul>
             <li><a href="#gaudi">Barcelona's
             ➝ Architect</a></li>
             <li><a href="#sagrada-familia"
             ➝ lang="es">La Sagrada Família
             ➝ </a></li>
             <li><a href="#park-guell">Park
             ➝ Guell</a></li>
          </ul>
       </nav>
    </header>

    <main role="main">
       <article class="architect gaudi">
          <h1 id="gaudi">Barcelona's
          ➝ Architect</h1>

          <p>Antoni Gaudí's incredible
          ➝ buildings...</p>
          ...

          <h2 id="sagrada-familia" lang=
          ➝ "es">La Sagrada Família</h2>
          ...

          <h2 id="park-guell">Park Guell
          ➝ </h2>
          ...
       </article>
    </main>
    ...
</div>
</body>
</html>
```

图 3.14.1　nav 中的链接指向 h1 和 h2 中的 id。为一个或多个元素添加 class 属性，就可以对所有这类元素统一进行格式化。例如，可以将 architect 类应用到关于其他建筑师的内容，从而为它们应用一致的格式；gaudi 类可以为与他相关的内容提供另一种样式

1. 为元素添加唯一的 ID

在元素的开始标签中输入 id="*name*"，其中 *name* 是唯一标识该元素的名称（图 3.14.1）。*name* 几乎可以是任何字符，只要不以数字开头且不包含空格。

2. 为元素指定类别的方法

在元素的开始标签中输入 class="*name*"，其中 *name* 是类别的名称（图 3.14.1）。如果要指定多个类别，用空格将不同的类别名称分开即可，如 class="*name anothername*"。（可以指定两个以上的类别名称。）

提 示　关于通过 id 或 class 为元素添加样式，参见 9.3 节。不过，需要在这里说明的是，推荐使用类为元素添加样式。

提 示　关于创建指向标有 id 的元素的链接，参见 6.2 节。如何对具有 id 或 class 的元素使用 JavaScript 已经超出了本书的范围，不过我们在第 19 章会看几个示例。

提 示　HTML 文档中的每个 id 都必须是唯一的。换句话说，一个页面里不能出现两个具有相同 id 的元素，并且每个元素都只能有一个 id。相同的 id 可以出现在不同的页面里，同一 id 也不一定每次都赋给同一元素，尽管这是惯常的做法。

提 示　相反，一个 class 名称可以分配给页面中任意数量的元素，并且一个元素可以有一个以上的 class。

提 示　class 和 id 属性可以应用于任何 HTML 元素。元素可以同时拥有 id 和任意数量的 class。

提 示　在 class 和 id 名称中，通常使用短横线分隔多个单词，例如 class="footer-page"。

提示　不管你打算如何使用 id 和 class，都应该为它们选择有意义的名称。例如，如果你使用 class 是出于格式化目的，应避免使用描述表现样式的名称，如 class="red"，因为你可能在下周决定将配色方案改为蓝色。尽管在 CSS 中对分配给某一类元素的颜色进行修改是相当容易的，但这样做会导致你的 HTML 拥有一个名为红色却实际呈现为另一种颜色的 class。同时，改变 HTML 中所有的 class 通常是一项繁琐的工作。这点在你开始学习 CSS 之后尤为明显。

提示　可以使用 class 属性实现微格式（microformat）。更多信息参见 http://microformats.org。

3.15　为元素添加 title 属性

可以使用 title 属性（不要与 title 元素混淆）为网站上任何部分加上提示标签（参见图 3.15.1 至图 3.15.3）。不过，它们并不只是提示标签，加上它们之后屏幕阅读器可以为用户朗读 title 文本，因此使用 title 可以提升无障碍访问功能。

```
…
<ul title="Table of Contents">
    <li><a href="#gaudi" title="Learn about
    → Antoni Gaudí">Barcelona's Architect
    → </a></li>
    <li><a href="#sagrada-familia" lang="es">
    → La Sagrada Família</a></li>
    <li><a href="#park-guell">Park Guell</a>
    → </li>
</ul>
…
</body>
</html>
```

图 3.15.1　可以为任何元素添加 title，不过用的最多的是链接

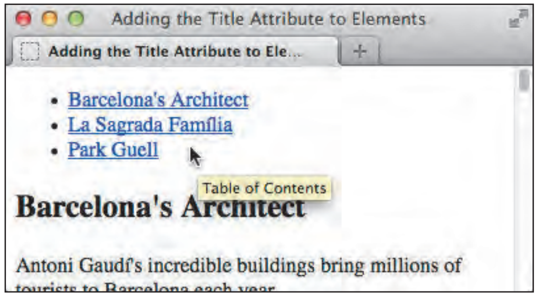


图 3.15.2　当访问者将鼠标指向加了说明标签的元素时，就会显示 title。如果指向 Barcelona's Architect 链接……

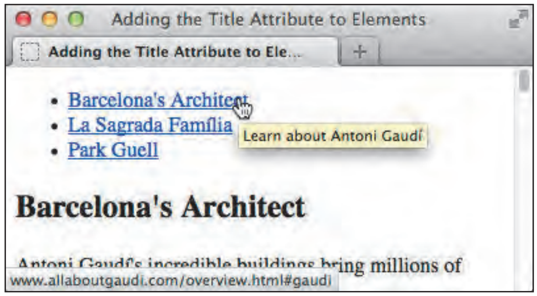


图 3.15.3　……会看到 Learn about Antoni Gaudí，因为该链接有 title 属性

在网页中为元素添加标签

在要添加 title 的 HTML 元素中，输入 title="*label*"，其中 *label* 是访问者将鼠标移到这个元素上时希望出现在提示框里的文本，或者希望由屏幕阅读器朗读的文本。

提示　旧版本的 Internet Explorer（IE7 及之前的版本）还会将 img 元素（参见第 5 章）的 alt 属性作为提示框的文本。不过，如果 img 元素同时包括 title 和 alt 属性，则提示框会采用 title 属性的内容，而不是 alt 属性的内容。

3.16　添加注释

可以在 HTML 文档中添加注释，标明区块开始和结束的位置，提醒自己（或未来的代码编辑者）某段代码的意图，或者阻止内容显示等，如图 3.16.1 所示。这些注释只会在用文本编辑器或浏览器的“查看源代码”选项打开文档时显示出来。访问者在浏览器中是看不到它们的，如图 3.16.2 所示。

```
...

<!-- ==== 开始主体内容 ==== -->
<main role="main">
    <article class="architect">
        <h1 id="gaudi">Barcelona's Architect
        → </h1>

        <!-- 这一段不会显示出来，因为它被注释
        → 掉了
        <p>Antoni Gaudí's incredible
        → buildings bring millions of
        → tourists to Barcelona each year.
        → </p>
        -->

        <p>Gaudí's non-conformity, already
        → visible in his teenage years...</p>
        ...
    </article>
</main>
<!-- 结束主体内容 -->

<!-- ==== 开始附注栏 ==== -->
... [附注栏内容] ...
<!-- 结束附注栏 -->

...
```

图 3.16.1　这段示例代码包括五个注释。其中有四个一起标记了两个区块的开始和结束位置。另一个“注释掉”了第一段，这样它就不会显示在页面中（如果希望永久性地移除该段，最好将它从 HTML 中删除）

在 HTML 页面中添加注释的步骤

(1) 在 HTML 文档中希望插入注释的位置，输入 <!--。

(2) 输入注释。

(3) 输入 --> 结束注释文本。



图 3.16.2　注释是不可见的。类似地，如果将一些内容“注释掉”，它们也不会显示出来。这段代码中的第一个段落并不会显示在浏览器中

提　示　在主要区块的开头和结尾处添加注释是一种常见的做法，这样可以让你或一起合作的开发人员将来修改代码变得更加容易。笔者喜欢使用一种比结束注释更为醒目的格式编写开始注释，从而更容易区分这两个位置。

提　示　在发布网站之前，应该用浏览器查看一下加了注释的页面。这样能帮你避免由于弄错注释格式导致私人化的注释内容直接暴露给公众访问者的情况。

提　示　对特别私人化的注释要格外小心。尽管通常通过浏览器访问页面不会看见注释内容，但通过浏览器的“查看源代码”功能就能看见它们，如果用户将网页保存为 HTML 源代码，他们也能看见这些注释。

提　示　注释不能嵌套在其他注释里。

第 4 章

文本

4

本章内容

- 添加段落
- 指定细则
- 标记重要或强调的文本
- 创建图
- 指明引用或参考
- 引述文本
- 指定时间
- 解释缩写词
- 定义术语
- 创建上标和下标
- 添加作者联系信息
- 标注编辑和不准确的文本
- 标记代码
- 使用预格式化的文本
- 突出显示文本
- 创建换行
- 创建 span
- 其他元素

除非网站添加了太多视频和图片，否则网页的大部分内容还是文本。本章将说明针对不同的文本类型，尤其是（但不仅仅是）那些句子或短语里的文本，应该选择哪些合适的 HTML 语义化元素。

例如，em 元素用于标识强调的文本，cite 元素用于标识对艺术作品、电影、图书等内容的引用。

浏览器通常会为这些文本元素添加不同的样式，以区别于普通文本。例如，em 和 cite 元素中的文本都是斜体的。又如，code 元素专门用来格式化脚本或程序中的代码，该元素中的文本默认使用等宽字体。

内容显示的样子与为其使用的标记没有关系。因此，不应该为了让文字变为斜体就使用 em 或 cite，添加样式是 CSS 的事情。

相反，应该选择能描述内容的 HTML 元素。如果浏览器默认添加的样式与你想用 CSS 设置的样式相同，那只不过是一种额外的奖励。如果不相同，直接自己编写 CSS 覆盖默认样式就可以了。

4.1　添加段落

HTML 会忽略你在文本编辑器中输入的回车符和其他额外的空格。要在网页中开始一个新的段落，应该使用 p 元素，如图 4.1.1 和图 4.1.2 所示。

```
...
<body>

<h1>Antoni Gaudí</h1>
<p>Many tourists are drawn to Barcelona
→ to see Antoni Gaudí's incredible
→ architecture.</p>

<p>Barcelona celebrated the 150th
→ anniversary of Gaudí's birth in
→ 2002.</p>

<h2 lang="es">La Casa Milà</h2>
<p>Gaudí's work was essentially useful.
→ <span lang="es">La Casa Milà</span> is
→ an apartment building and real people
→ live there.</p>

<h2 lang="es">La Sagrada Família</h2>
<p>The complicatedly named and curiously
→ unfinished Expiatory Temple of the
→ Sacred Family is the most visited
→ building in Barcelona.</p>

</body>
</html>
```

图 4.1.1　毫无疑问，p 是最常使用的 HTML 元素之一（注意：在实践中，我通常会使用 article 包住这些内容。这里省略了该标记，是为了让例子显得更为通用，同时避免产生 p 元素必须嵌套在 article 中的印象）



图 4.1.2　这是段落通常默认的呈现方式。默认情况下，浏览器会在标题和段落之间，以及不同的段落之间添加垂直间距。使用 CSS 可以控制所有内容元素的格式

创建新段落的步骤

(1) 输入 <p>。

(2) 输入新段落的内容。

(3) 输入 </p> 结束这个段落。

提 示　可以为段落（以及页面上的其他文本）添加样式，包括字体、字号、颜色等。详细内容参见第 10 章。

提 示　要控制段内行间距，参见 10.7 节。要控制段前段后的间距，参见 11.8 节和 11.10 节。

提 示　可以通过 CSS 改变段落文本的对齐方式，包括左对齐、右对齐和居中对齐（参见 10.13 节）。

4.2　指定细则

根据 HTML5，small 表示细则一类的旁注（side comment），“通常包括免责声明、注意事项、法律限制、版权信息等。有时我们还可以用它来表示署名，或者满足许可要求。”

small 通常是行内文本中的一小块，而不是包含多个段落或其他元素的大块文本，参见图 4.2.1 和图 4.2.2。

指定细则的步骤

(1) 输入 <small>。

(2) 输入表示免责声明、注解、署名等类型的文本。

(3) 输入 </small>。

提 示　一些浏览器会将 small 包含的文本显示为小字号（图 4.2.2）。不过，一定要在符合内容语义的情况下使用该元素，而不是为了减小字号而使用。总是可以用 CSS 控制字号（如果你想，甚至可以让 small 包含的文本的字号变大），更多信息参见 10.6 节。

```
...
<body>

<p>Order now to receive free shipping.
<small>(Some restrictions may apply.)
→ </small></p>

...

<footer role="contentinfo">
    <p><small>&copy; 2013 The Super
    → Store. All Rights Reserved.
    → </small></p>
</footer>

</body>
</html>
```

图 4.2.1　在下面的两个例子中，我们用 small 元素来表示简短的法律声明。在第二个例子中，small 表示的是包含在页面级 footer 里的版权声明，这是一种常见的用法



图 4.2.2　在一些浏览器中，small 元素中文本的字号会比普通文本的字号小，不过，视觉上的大小与是否该用它标记内容毫无关系

提 示　用 small 标记页面的版权信息是一种常见的做法（图 4.2.1、图 4.2.2）。不过，small 只适用于短语，因此不要用它标记长的法律声明，如“使用条款”和“隐私政策”页面。根据需要，应该用段落或其他语义标签标记这些内容。

4.3　标记重要和强调的文本

strong 元素表示内容的重要性，而 em 则表示内容的着重点。根据内容需要，这两个元素既可以单独使用，也可以一起使用，参见图 4.3.1 和图 4.3.2。

```
...
<body>

<p><strong>Warning: Do not approach the
→ zombies <em>under any circumstances</em>
→ </strong>. They may <em>look</em>
→ friendly, but that's just because they want
→ to eat your arm.</p>

</body>
</html>
```

图 4.3.1　第一个句子既有 strong 又有 em，而第二个句子只有 em

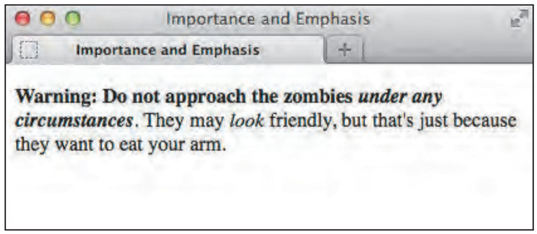


图 4.3.2　浏览器通常将 strong 文本以粗体显示，而将 em 文本以斜体显示。如果 em 是 strong 的子元素（如图 4.3.1 中的第一个句子），将同时以斜体和粗体显示文本

标记重要文本

(1) 输入 。

(2) 输入想标记为重要内容的文本。

(3) 输入 。

强调文本

(1) 输入 。

(2) 输入想强调的文本。

(3) 输入 。

提示　不要使用 b 元素代替 strong，也不要使用 i 元素代替 em。尽管它们在浏览器中显示的样式是一样的，但它们的含义却很不一样（参见补充材料“HTML5 中重新定义的 b 和 i 元素”）。

提示　就像在说话时强调某些词语一样，em 在句子中的位置会影响句子的含义。例如，<p>Run over here.</p> 和 <p>Run over here.</p> 表达的意思是不一样的。

提示　可以在标记为 strong 的短语中再嵌套 strong 文本。如果这样做，作为另一个 strong 的子元素的 strong 文本的重要程度会递增。这种规则对嵌套在另一个 em 里的 em 文本也适用。例如，在语句 <p>Remember that entries are **due by March 12th**.</p> 中，“due by March 12th”要比其他 strong 文本更为重要。

提示　可以用 CSS 将任何文本变为粗体或斜体，也可以覆盖 strong 和 em 等元素的浏览器默认显示样式（图 4.3.2）。更多细节参见 10.4 节和 10.5 节。

提示　如果你有过编写旧版本 HTML 的经验，你或许知道，那时 strong 所表示文本的强调程度比 em 表示的文本要高。不过，在 HTML5 中，em 是表示强调的唯一元素，而 strong 表示的则是重要程度。

HTML5 中重新定义的 b 和 i 元素

HTML5 强调元素的语义，而非表现。b 和 i 元素是早期 HTML 遗留下来的产物，它们分别用于将文本变为粗体和斜体（那时 CSS 还未出现）。HTML 4 和 XHTML 1 当然抛弃了它们，因为它们本质上是用于表现的。当时的规范建议编码人员用 strong 替代 b，用 em 替代 i。不过，事实证明，em 和 strong 有时在语义上并不合适。为此，HTML5 重新定义了 b 和 i。

传统出版业里的某些排版规则在现有的 HTML 语义中还找不到对应物，其中就包括用斜体表示植物学名（如“*Ulmus americana* is the Massachusetts state tree.”）、具体的交通工具名称（如“We rode the *Orient Express*.”）及外来语（如“The couple exhibited a *joie de vivre* that was infectious.”）。这些词语不是为了强调而加上斜体的，只是样式上的惯例。

为了应对这些情况，HTML5 没有创建一些新的语义化元素（进一步把事情搞复杂），而是采取了一种很实际的做法，直接利用现有元素：em 用于所有层次的强调，strong 用于表示重要性，而其他情况则使用 b 和 i。

这意味着，尽管 b 和 i 并不包含任何明显的语义，但读者仍能发现它们与周边文字的差别。而且你还可以通过 CSS 改变它们粗体或斜体的样式。HTML5 强调，b 和 i 应该是其他元素（如 strong、em、cite 等）都不适用时的最后选择。

◎ b 元素简介

HTML5 将 b 重新定义为：

b 元素表示出于实用目的提醒读者注意的一块文字，不传达任何额外的重要性，也不表示其他的语态和语气，用于如文档摘要里的关键词、评论中的产品名、基于文本的交互式软件中指示操作的文字、文章导语等。

例如：

```
<p>The <b>XR-5</b>, also dubbed the <b>Extreme Robot 5</b>, is the best robot we've ever
➝ tested.</p>
```

b 元素默认显示为粗体。

◎ i 元素简介

HTML5 将 i 重新定义为：

i 元素表示一块不同于其他文字的文字，具有不同的语态或语气，或其他不同于常规之处，用于如分类名称、技术术语、外语里的惯用语、翻译的散文、西方文字中的船舶名称等。

例如：

```
<p>The <i lang="la">Ulmus americana</i> is the Massachusetts state tree.</p>
<p>We rode the <i>Orient Express</i>.<p>
<p>The couple exhibited a <i lang="fr">joie de vivre</i> that was infectious.<p>
```

i 元素默认显示为斜体。

4.4 创建图

你一定在报纸、杂志、报告等其他媒介上看到过图。通常，图是由页面上的文本引述出来的。本书的大多数页面都有这样的例子。

在 HTML5 出现之前，没有专门实现这个目的的元素，因此一些开发人员想出了他们自己的解决办法（通常会使用不那么理想的、没有语义的 div 元素）。通过引入 figure 和 figcaption，HTML5 改变了这种情况（参见图 4.4.1 和图 4.4.2）。图可以是图表、照片、图形、插图、代码片段，以及其他类似的独立内容。

可以由页面上的其他内容引出 figure（如图 4.4.1 和图 4.4.2 所示）。figcaption 是 figure 的标题，可选，出现在 figure 内容的开头或结尾处。

创建图及其标题的步骤

(1) 输入 <figure>。

(2) 作为可选步骤，输入 <figcaption> 开始图的标题。

(3) 输入标题文字。

(4) 如果在第 (2)、(3) 步创建了标题，就输入 </figcaption>。

(5) 添加图像、视频、数据表格等内容的

代码创建图。

(6) 如果没有在 figure 内容之前包含 figcaption，则可以在内容之后重复第 (2) ~ (4) 步。

(7) 输入 </figure>。

```
...
<body>
...
<article>
    <h1>2013 Revenue by Industry</h1>

    <p>... [报告内容] ...</p>

    <figure>
       <figcaption><b>Figure 3:</b>
       → Breakdown of Revenue by
       → Industry</figcaption>

       <img src="chart-revenue.png"
       → width="180" height="143" alt=
       → "Revenue chart: Clothing 42%,
       → Toys 36%, Food 22%" />
    </figure>

    <p>As Figure 3 illustrates, ... </p>

    <p>... [其他报告内容] ...</p>
</article>
...
</body>
</html>
```

图 4.4.1　这个 figure 只有一个图表，不过放置多个图像或其他类型的内容（如数据表格、视频等）也是允许的。figcaption 元素并不是必需的，但如果包含它，它就必须是 figure 元素内嵌的第一个或最后一个元素。除了在现代浏览器中从新的一行开始显示，figure 没有默认样式，如图 4.4.2 所示（注意：figure 元素并不一定要包含在 article 里，但在大多数情况下这样做比较合适）

提 示　通常，figure 是引用它的内容的一部分（图 4.4.1），但它也可以位于页面的其他部分，或位于其他页面（如附录）。

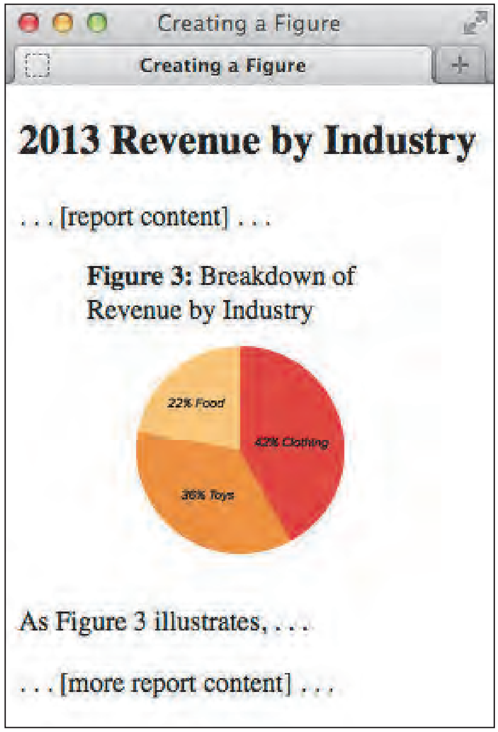


图 4.4.2　这个包含图表及其标题的 figure 出现在 article 文本中间。图以缩进的形式显示，这是浏览器的默认样式（参见最后一条提示）

提 示　figure 元素可以包含多个内容块。例如，图 4.4.1 中可以包含两个图表：一个表示收入，一个表示利润。不过要记住，不管 figure 里有多少内容，只允许有一个 figcaption。

提 示　不要简单地将 figure 作为在文本中嵌入独立内容实例的方法。这种情况下，通常更适合用 aside 元素（参见 3.10 节）。

提 示　要了解如何结合使用 blockquote 和 figure 元素，参见 4.6 节。

提示 可选的 figcaption 必须与其他内容一起包含在 figure 里面，不能单独出现在其他位置。

提示 figcaption 中的文本不必以 Figure 3、Exhibit B 等字样开头，对内容的一句简短描述即可，就像照片的描述文本。

提示 现代浏览器在默认情况下会为 figure 添加 40px 宽的左右外边距（图 4.4.3）。可以使用 CSS 的 margin-left 和 margin-right 属性修改这一样式。例如，使用 margin-left: 0; 可以让图直接抵到页面左边缘。还可以使用 figure { float: left; } 让包含 figure 的文本环绕在它周围（这样，文本就会围在图的右侧）。可能还需要为 figure 设置 width，使之不至于占据太大的水平空间。本书将从第 7 章开始阐述 CSS，float 和 width 属性会在第 11 章中讲到。

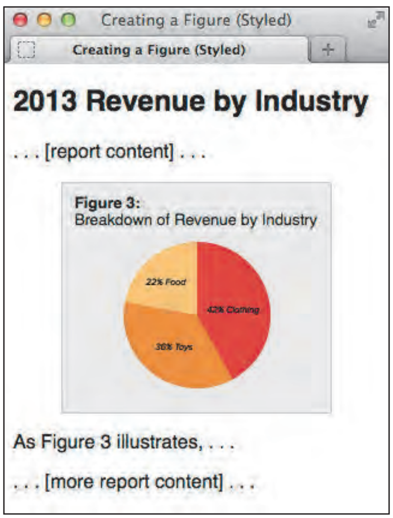


图 4.4.3 使用一点 CSS 就可以让 figure 与它旁边的文本区分开来。这个简单的例子见 www.htmlcssvqs.com/8ed/figure-styled/

4.5 指明引用或参考

使用 cite 元素可以指明对某内容源的引用或参考。例如，戏剧、脚本或图书的标题，歌曲、电影、照片或雕塑的名称，演唱会或音乐会，规范、报纸或法律文件等（参见图 4.5.1 和图 4.5.2）。

```
...
<body>

<p>He listened to <cite>Abbey Road</cite>
→ while watching <cite>A Hard Day's Night
→ </cite> and reading <cite>The Beatles
→ Anthology</cite>.

<p>When he went to The Louvre, he learned
→ that <cite>Mona Lisa</cite> is also
→ known as <cite lang="it">La Gioconda
→ </cite>.</p>

</body>
</html>
```

图 4.5.1 在这个例子中，cite 元素标记的是音乐专辑、电影、图书和艺术作品的标题（注意：最后一个例子中的 lang="it" 表明 cite 文本的语言是意大利语）

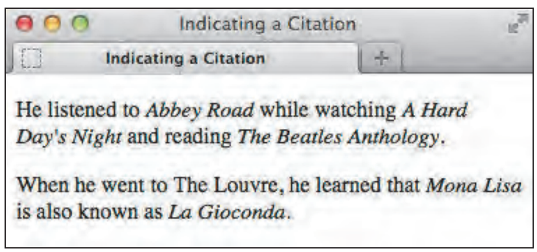


图 4.5.2 cite 元素默认以斜体显示

引用参考的步骤

(1) 输入 <cite>。

(2) 输入参考的名称。

(3) 输入 </cite>。

提示 对于要从引用来源中引述内容的情况，使用 blockquote 或 q 元素标记引述的文本（参见 4.6 节）。要弄清楚的是，cite 只用于参考源本身，而不是从中引述的内容。

HTML5 与使用 cite 元素表示名称

HTML5 声明，不应使用 cite 作为对人名的引用（这是开发社区众多意见分歧中的一个），但 HTML 以前的版本允许这样做，而且很多设计和开发人员仍在这样做。

HTML4 的规范有以下例子（已将元素名称由大写字母改为小写）：

```
As <cite>Harry S. Truman</cite> said,
<q lang="en-us">The buck stops here.</q>
```

除了这些例子，有的网站经常用 cite 标记在博客和文章中发表评论的访问者的名字（WordPress 的默认主题就是这样做的）。

很多开发人员表示他们将继续对与页面中的引文有关的名称使用 cite，因为 HTML5 没有提供他们认为可接受的其他元素（即 span 和 b 元素）。Jeremy Keith 提供了一个很好的例子，见 http://24ways.org/2009/incite-a-riot/。

4.6　引述文本

有两个特殊的元素用以标记引述的文本。blockquote 元素表示单独存在的引述（通常更长，但也可能不是），参见图 4.6.1 和图 4.6.2，它默认显示在新的一行，如图 4.6.3 所示。而 q 元素则用于短的引述，如句子里面的引述（图 4.6.4）。

```
...
<body>

<p>He especially enjoyed this selection from
➝ <cite>The Adventures of Huckleberry Finn
➝ </cite> by Mark Twain:</p>

<blockquote cite="http://www.
➝ marktwainbooks.edu/the-adventures-of-
➝ huckleberry-finn/">
    <p>We said there warn't no home like a
    ➝ raft, after all. Other places do seem
    ➝ so cramped up and smothery, but a
    ➝ raft don't. You feel mighty free and
    ➝ easy and comfortable on a raft.</p>
</blockquote>

<p>It reminded him of his own youth exploring
➝ the county by river in the summertime.</p>

</body>
</html>
```

图 4.6.1　根据需要，blockquote 可长可短。可以包含 cite 属性（不要与第一段中出现的 cite 元素混淆）提供引述文本的位置

```
...

<figure>
    <blockquote>
    I want all my senses engaged. Let
    ➝ me absorb the world's variety and
    ➝ uniqueness.
    </blockquote>
    <figcaption>— Maya Angelou</figcaption>
</figure>

...
```

图 4.6.2　如果要添加署名，署名应该放在 blockquote 外面。可以把署名放在 p 里面，不过使用 figure 和 figcaption 可以更好地将引述文本与其来源关联起来，如下所示（参见 4.5 节）

浏览器应对 q 元素中的文本自动加上特定语言的引号，但不同浏览器的效果并不相同（参见图 4.6.5）。一定要阅读本节提示，了解 q 元素的替代方法。

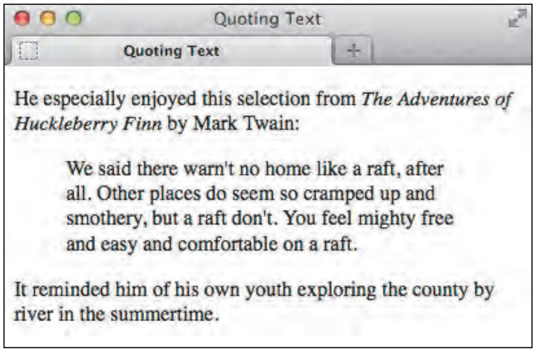


图 4.6.3　浏览器默认对 blockquote 文本进行缩进，cite 属性的值则不会显示出来（对此，参见第二条提示中的推荐做法）。不过，所有的浏览器都支持 cite 元素，通常对其中的文本以斜体显示，如图所示。所有这些默认样式都可以被 CSS 覆盖

```
...
<body>

<p>And then she said, <q>Have you read
→ Barbara Kingsolver's <cite>High Tide in
→ Tucson</cite>? It's inspiring.</q></p>

<p>She tried again, this time in French:
→ <q lang="fr">Avez-vous lu le livre <cite
→ lang="en">High Tide in Tucson</cite> de
→ Kingsolver? C'est inspirational.</q></p>

</body>
</html>
```

图 4.6.4　这里是两个 q 的例子。如果引述文本的语言与页面默认语言（通过 html 元素的 lang 属性指定）不同，就在 q 元素中加上 lang 属性

1. 引述块级文本的步骤

(1) 输入 <blockquote 开始一个块级引述。

(2) 如果需要，输入 cite="*url*"，其中 *url* 为引述来源的地址。

(3) 输入 > 以结束开始标签。

(4) 输入希望引述的文本，并用段落等适当的元素包围。

(5) 输入 </blockquote>。

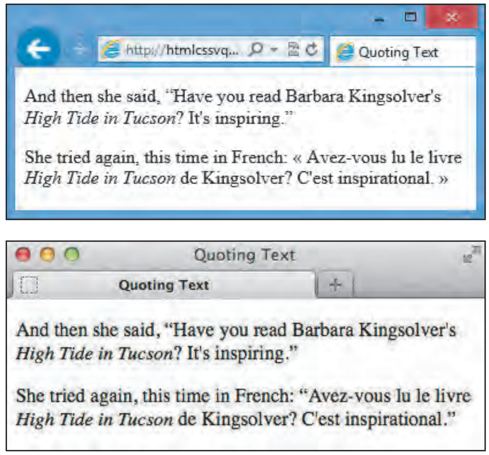


图 4.6.5　浏览器应该自动在 q 元素周围加上特定语言的引号。在这个例子中，这意味着英文和法文分别使用不同的引号。IE（上图）和 Chrome 的处理是正确的。Firefox（下图）对英文的处理是正确的，而法文则不正确。Opera 和 Safari 对两种语言的处理都不正确，它们都显示直引号（包括法文）。这样的不一致影响了 q 元素的有效性

2. 引述行内文本的步骤

(1) 输入 <q 开始引述字词或短语。

(2) 如果需要，输入 cite="*url*"，其中 *url* 为引述来源的地址。

(3) 如果引述内容的语言与页面默认语言（通过 html 元素的 lang 属性指定）不同，输入 lang="*xx*"，其中 *xx* 是引述内容语言的两个字母的代码。这个代码用于判断需要使用的引号的类型（英语为 " "，而很多欧洲语言则为 «»），但浏览器对引号的呈现方式可能不同。

(4) 输入 > 以结束开始标签。

(5) 输入要引述的文本。

(6) 输入 </q>。

提示　如果 blockquote 中仅包含一个单独的段落或短语，可以不必将其包在 p 中再放入 blockquote。

提示　可以对 blockquote 和 q 使用可选的 cite 属性，提供引述内容来源的 URL。尽管浏览器通常不会将 cite 的 URL 呈现给用户（参见图 4.6.3），但理论上讲，该属性对搜索引擎或其他收集引述文本及其引用的自动化工具来说还是有用的。如果要让访问者看到这个 URL，可以在内容中使用链接（a 元素）重复这个 URL。也可以使用 JavaScript 将 cite 的值暴露出来（可以从网上搜到示例代码），但这样做的效果稍差一些。

提示　q 元素引用的内容不能跨越不同的段落，在这种情况下应使用 blockquote。

提示　不要仅仅因为需要在字词两端添加引号就使用 q。例如，<p>Every time I hear the word <q>soy</q>, I jump for joy.</p> 是不正确的，因为 soy 并不是对某参考源的引用。

提示　blockquote 和 q 元素可以嵌套。例如，<p>The short story began, <q>When she was a child, she would say, <q>Howdy, stranger!</q> to everyone she passed.</q> </p>。嵌套的 q 元素应该自动加上正确的引号（例如，在英语中外面的是双引号，里面的是单引号）。由于内外引号在不同语言中的处理方式不一样，因此要根据需要在 q 元素中加上 lang 属性（参见图 4.6.4）。不幸的是，浏览器对嵌套 q 元素的支持程度并不相同，其实浏览器对非嵌套 q 元素的支持也不同（参见图 4.6.5）。

提示　由于 q 元素的跨浏览器问题（参见图 4.6.5），很多（可能是大多数）开发人员避免使用 q 元素，而是选择直接输入正确的引号或使用字符实体。

4.7　指定时间

我们可以使用 time 元素标记时间、日期或时间段，这是 HTML5 新增的元素。呈现这些信息的方式有多种，参见图 4.7.1 至图 4.7.3。

```
...
<body>

<p>The train arrives at <time>08:45</time>
➝ and <time>16:20</time> on <time>
➝ 2017-03-19</time>.</p>

<p>They made their dinner reservation for
➝ <time datetime="2013-11-20T18:30:00">
➝ tonight at 6:30</time>.</p>

<p>We began our descent from the peak of
➝ Everest on <time datetime="1952-06-12T
➝ 11:05:00">June 12, 1952 at 11:05 a.m.
➝ </time></p>

<p>The film festival is <time datetime=
➝ "2014-07-13">July 13</time>-<time
➝ datetime="2014-07-16">16</time>.</p>

<!-- 不包含年份的例子 -->
<p>Her birthday is <time datetime="03-29">
➝ March 29th</time>.</p>

<!-- 时间段的例子 -->
<p>The meeting lasted <time>2h 41m 3s
➝ </time> instead of the scheduled <time
➝ datetime="2h 30m">two hours and thirty
➝ minutes</time>.</p>

</body>
</html>
```

图 4.7.1　如第一个例子所示，time 元素最简单的用法是不包含 datetime 属性。在忽略 datetime 属性的情况下，它们的确提供了具备有效的机器可读格式的时间和日期。在其余的例子中，由于提供了 datetime 属性，time 标签中的文本并未严格使用有效的格式

time 中包含的文本内容（即 <time>text </time>）会出现在屏幕上，对用户可见（参见图 4.7.2 和图 4.7.4），而可选的 datetime

属性则是为机器准备的。该属性需要遵循特定的格式，本节后面的补充材料“理解有效的时间格式”说明了相关的基本知识，第一条提示则说明了另一种格式要求的特定情形。

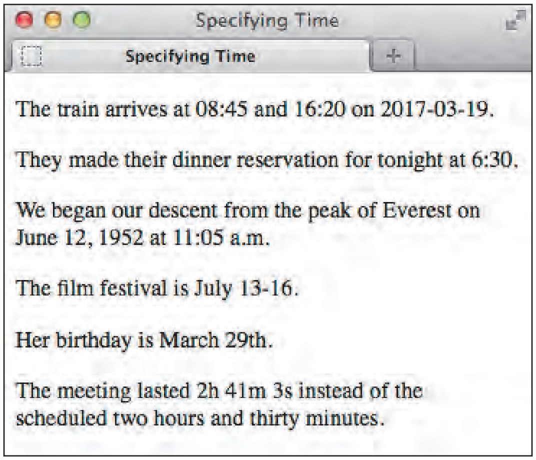


图 4.7.2 浏览器只显示 time 元素的文本内容，而不会显示 datetime 的值

```
...
<body>

<article>
    <h1>Cheetah and Gazelle Make Fast
    → Friends</h1>
    <p><time datetime="2014-10-15">October
    → 15, 2014</time></p>

    ... [文章内容] ...
</article>

</body>
</html>
```

图 4.7.3 这里展示了如何在博客或新闻文章中包含日期。datetime 属性中的值表示的是文本内容的机器可读格式

指定准确时间、日期或时间段的步骤

(1) 输入 <time 开始 time 元素。

(2) 如果需要，输入 datetime="time"，其中 time 表示的是第 (4) 步中文本的机器可读格式(参见补充材料“理解有效的时间格式”)。

(3) 输入 > 结束开始标签。

(4) 输入要显示在浏览器中的表示时间、日期或时间段的文本（如果第 (2) 步中没有包含 datetime 属性，参见第一条提示）。

(5) 输入 </time>。

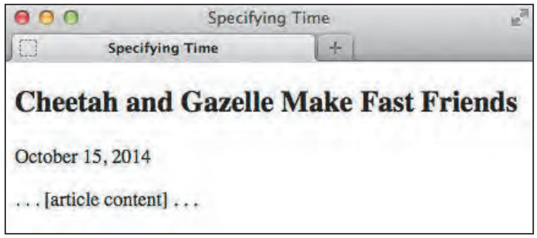


图 4.7.4 恰如预期，日期显示在标题的下方

提示 如果忽略 datetime 属性，文本内容就必须是合法的日期或时间格式。也就是说，图 4.7.1 中的第一个例子不能写成 <p>The train arrives at <time>8:45 a.m.</time> and <time>4:20 p.m.</time> on <time>April 20th, 2015</time>.</p>，因为这三处 time 文本的格式都不正确。不过，如果包含了 datetime，文本内容就可以按你希望的任何形式表示日期、时间或时间段了，就像图 4.7.1 中的其他例子那样。

提示 datetime 属性不会单独产生任何效果，但它可以用于在 Web 应用（如日历应用）之间同步日期和时间。这就是必须使用标准的机器可读格式的原因，这样，程序之间就可以使用相同的“语言”来共享信息。

提示 不能在 time 元素中嵌套另一个 time 元素，也不能在没有 datetime 属性的 time 元素中包含其他元素（只能包含文本）。

提示　在早期的 HTML5 说明中，time 元素可以包含一个名为 pubdate 的可选属性（要知道，HTML 这门语言还在不断地向前发展）。不过，pubdate 已不再是 HTML5 的一部分。在这里提到这个属性是为了让你偶然在较早的教程或书（如本书的第 7 版）中碰到该属性时不必犹豫是否应该使用它（当然不应该使用）。

理解有效的时间格式

datetime 属性（或者没有 datetime 属性的 time 元素）必须提供特定的机器可读格式的日期和时间。这可以简化为下面的形式：

```
YYYY-MM-DDThh:mm:ss
```

例如（当地时间）：

```
1985-11-03T17:19:10
```

表示“当地时间 1985 年 11 月 3 日下午 5 时 19 分 10 秒”。小时部分使用 24 小时制，因此表示下午 5 点应使用 17 而非 05。如果包含时间，秒是可选的。（也可以使用 hh:mm.sss 格式提供时间的毫秒数。注意毫秒数之前的符号是一个点。）

如果要表示时间段，则格式稍有不同。有好几种语法，不过最简单的形式为：

*n*h *n*m *n*s

（其中，三个 *n* 分别表示小时数、分钟数和秒数。）

图 4.7.1 中的最后一个例子就是表示时间段的实例。

◎ 全球日期和时间及时差

你也可以将日期和时间表示为世界时。在末尾加上字母 Z，就成了 UTC（Coordinated Universal Time，全球标准时间）。UTC 是主要的全球时间标准。（参见 https://en.wikipedia.org/wiki/Coordinated_Universal_Time。）

例如（使用 UTC 的世界时）：

```
1985-11-03T17:19:10Z
```

也可以通过相对 UTC 时差的方式表示时间。这时不写字母 Z，写上 -（减）或 +（加）及时差即可。

例如（含相对 UTC 时差的世界时）：

```
1985-11-03T17:19:10-03:30
```

表示“纽芬兰标准时（NST）1985 年 11 月 3 日下午 5 时 19 分 10 秒”（NST 比 UTC 晚 3 个半小时）。UTC 时区列表参见 http://en.wikipedia.org/wiki/List_of_time_zones_by_UTC_offset。

提醒一下，如果确实要包含 datetime，也不必像图 4.7.1 中的示例那样提供时间的完整信息。

4.8 解释缩写词

缩写词很常见，如 Jr.、M.D.，甚至 good ol’HTML。可以使用 abbr 元素标记缩写词并解释其含义（图 4.8.1 至图 4.8.3）。不必对每个缩写词都使用 abbr，只在需要帮助访问者了解该词含义的时候使用。

```
...
<body>

<p>The <abbr title="National Football
→ League">NFL</abbr> promised a <abbr
→ title="light amplification by
→ stimulated emission of radiation">
→ laser</abbr> show at 9 p.m. after every
→ night game.</p>

<p>But, that's nothing compared to what
→ <abbr>MLB</abbr> (Major League
→ Baseball) did. They gave out free
→ <abbr title="self-contained underwater
→ breathing apparatus">scuba</abbr> gear
→ during rain delays.</p>

</body>
</html>
```

图 4.8.1 使用可选的 title 属性提供缩写词的全称。另外，也可以将全称放在缩写词后面的括号里（这样做或许更好）。还可以同时使用这两种方式，并使用一致的全称。对于像 laser（激光）、scuba（水中呼吸器）这样的词汇，大多数人都很熟悉了，因此其实并无必要对它们使用 abbr 并提供 title，这里只是用它们来演示示例

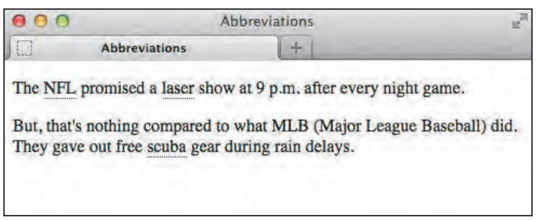


图 4.8.2 当缩写词有 title 属性时，Firefox 和 Opera 会添加虚线下划线以引起注意。可以通过 CSS 让其他浏览器也这样显示（参见提示）

解释缩写词的步骤

(1) 输入 <abbr。

(2) 作为可选的一步，输入 title="*expansion*"，其中 *expansion* 是缩写词的全称。

(3) 输入 >。

(4) 然后输入缩写词本身。

(5) 最后输入 </abbr> 结束标签。

(6) 作为可选的一步，输入一个空格和 (*expansion*)，其中 *expansion* 是缩写词的全称。

提示 通常，仅在缩写词第一次出现在屏幕上时给出其全称（通过 title 或括号的方式）。

提示 用括号提供缩写词的全称是解释缩写词最直接的方式，能让尽可能多的访问者看到这些内容（图 4.8.1）。例如，使用智能手机和平板电脑等触摸屏设备的用户可能无法移到 abbr 元素上查看 title 的提示框。因此，如果要提供缩写词的全称，应该尽量将它放在括号里。

提示 如果使用复数形式的缩写词，全称也要使用复数形式。

提示 作为对用户的视觉提示，Firefox 和 Opera 等浏览器会对带 title 的 abbr 文字使用虚线下划线（图 4.8.2）。想在其他浏览器中也这样显示，可以在样式表中加上这条语句：abbr[title] { border-bottom: 1px dotted #000; }。无论 abbr 是否添加了下划线样式，浏览器都会将 title 属性内容以提示框的形式显示出来（图 4.8.3）。

提示 如果看不到 abbr 有虚线下划线，试着为其父元素的 CSS 添加 line-height 属性（参见第 10 章）。

提 示　在 HTML5 之前有 acronym（首字母缩写词）元素，但设计和开发人员常常分不清楚缩写词和首字母缩写词，因此 HTML5 废除了 acronym 元素，让 abbr 适用于所有的场合。

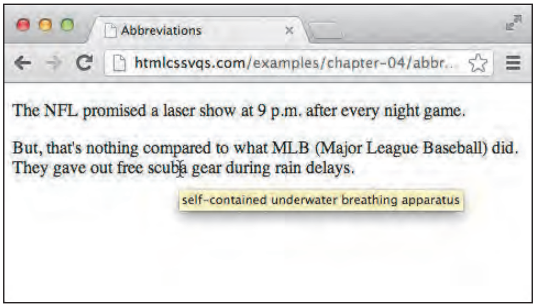


图 4.8.3　当访问者将鼠标移至 abbr 上，该元素 title 属性的内容就会显示在一个提示框里（这里还显示，在默认情况下，Chrome 等一些浏览器不会让带有 title 属性的缩写词跟普通文本有任何显示上的差别）

4.9　定义术语

在印刷物中，首次定义术语通常会对其添加区别于其他文本的格式（英文通常为斜体，汉语通常为黑体或者楷体），在其后提到术语时则不再使用特殊格式。

在 HTML 中定义术语时，可以使用 dfn 元素对其作**语义上**的区分。仅用 dfn 包围要定义的术语，而不是包围定义，如图 4.9.1 所示。同印刷物的惯例一样，后续使用术语时不再需要使用 dfn 对其进行标记，因为不再需要对其进行定义。（在 HTML 中，定义术语的地方称为“术语的定义实例”。）

标记术语的定义实例

(1) 输入 <dfn>。

(2) 输入要定义的术语。

(3) 输入 </dfn>。

```
...
<body>

<p>The contestant was asked to spell
➝ "pleonasm." She requested the definition
➝ and was told that <dfn>pleonasm</dfn>
➝ means "a redundant word or expression"
➝ (Ref: <cite><a href="http://dictionary.
➝ reference.com/browse/pleonasm" rel=
➝ "external">dictionary.com</a></cite>).</p>

</body>
</html>
```

图 4.9.1　注意，在这个例子中，尽管 *pleonasm* 出现了两次，但只对第二个用 dfn 进行了标记，因为那时才定义这个术语（即定义实例）。类似地，如果我在文档后面用到 pleonasm，也不会用 dfn 标记，因为我已经定义过它了。尽管浏览器在默认情况下会为 dfn 文本加上与普通文本不同的样式（参见图 4.9.2），但重要的是术语使用了特殊的标记。同时，不必在每次使用 dfn 时都使用 cite 元素，该元素只在引用参考源的时候使用

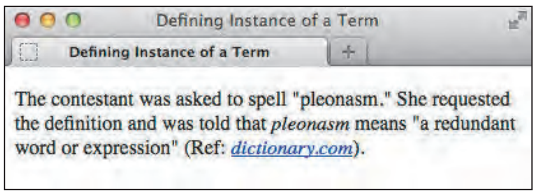


图 4.9.2　通常，dfn 元素默认以斜体显示，与 cite 一样

术语及其定义应位置相近

由 dfn 标记的术语与其定义的距离远近相当重要。如 HTML5 规范所述：“如果一个段落、描述列表或区块是某 dfn 元素距离最近的祖先，那么该段落、描述列表或区块必须包含该术语的定义。”简言之，dfn 元素及其定义必须挨在一起，否则便是错误的用法。图 4.9.1 和第四条提示中的例子都遵循这一规则——dfn 及其定义位于同一个段落。

提示 还可以在描述列表（dl 元素）中使用 dfn，参见第 15 章。

提示 仅在定义术语时使用 dfn，而不能为了让文字以斜体显示就使用该元素。使用 CSS 可以将任何文字变为斜体（参见 10.4 节）。

提示 dfn 可以在适当的情况下包住其他的短语元素，如 abbr。例如，<p>A **<dfn><abbr title="Junior">Jr.</abbr></dfn>** is a son with the same full name as his father.</p>。

提示 如果在 dfn 中添加可选的 title 属性，其值应与 dfn 术语一致。正如在上一条提示中看到的，如果只在 dfn 里嵌套一个单独的 abbr，dfn 本身没有文本，那么可选的 title 只能出现在 abbr 里。

4.10　创建上标和下标

比主体文本稍高或稍低的字母或数字分别称为上标和下标（参见图 4.10.1）。HTML 包含用来定义这两种文本的元素。常见的上标包括商标符号、指数和脚注编号等（参见图 4.10.2）；常见的下标包括化学符号等。

创建上标和下标的步骤

(1) 输入 <sub> 创建下标，或输入 <sup> 创建上标。

(2) 输入要出现在下标或上标里的字符或符号。

(3) 根据第 (1) 步中使用的元素，输入 </sub> 或 </sup> 结束该元素。

提示 大多数浏览器会自动将上标或下标文字的字号减少几磅。

```
...
<body>

<article>
    <h1>Famous Catalans</h1>
    <p>... Actually, Pablo Casals' real
    → name was <i>Pau</i> Casals, Pau
    → being the Catalan equivalent of Pablo
    → <a href="#footnote-1" title="Read
    → footnote 1"><sup>1</sup></a>.</p>

    <p>... Pau Casals is remembered in this
    → country for his empassioned speech
    → against nuclear proliferation at the
    → United Nations <a href="#footnote-2"
    → title="Read footnote 2"><sup>2</sup>
    → </a> ...</p>

    <footer>
        <p id="footnote-1"><sup>1</sup>It
        → means Paul in English.</p>
        <p id="footnote-2"><sup>2</sup>In
        → 1963, I believe.</p>
    </footer>
</article>

</body>
</html>
```

图 4.10.1　sup 元素的一种用法就是表示脚注编号。根据从属关系，我将脚注放在 article 的 footer 里，而不是整个页面的 footer 里。我还为文章中每个脚注编号创建了链接，指向 footer 内对应的脚注，从而让访问者更容易找到它们。同时，注意链接中的 title 属性也提供了一些提示

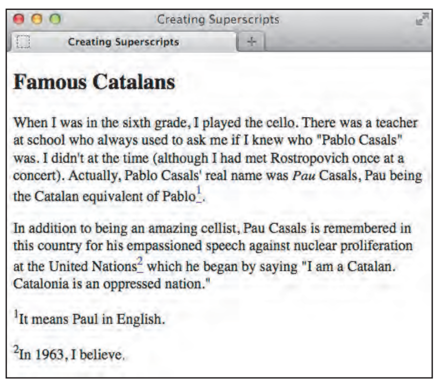


图 4.10.2　sup 元素的位置比同一行的文本要高。不过，不好的一点是，它们会扰乱行间距（参见最后一条提示）

提示 上标是对某些外语缩写词进行格式化的理想方式，例如，法语中用 Mlle 表示 Mademoiselle（小姐），西班牙语中用 3^{a} 表示 tercera（第三）。此外，一些数字形式也要用到上标，如 2nd、5th。

提示 下标适用于化学分子式（如 H_2O）。例如，`<p>I'm parched. Could I please have a glass of H<sub>2</sub>O?</p>.`。

提示 上标和下标字符会轻微地扰乱行与行之间的均匀的间距。例如，在图 4.10.2 中，第一段第 4 行和第 5 行之间的距离以及第二段第 2 行和第 3 行之间的距离比其他行间距要大一些。不过，可以使用一些 CSS 解决这个问题。解决的办法参见“修复使用 sub 和 sup 时的行间距问题”。

修复使用 sub 和 sup 时的行间距问题

sub 和 sup 元素会轻微地增大行高。幸好，用一点 CSS 就可以修复这个问题。下面的代码来自 Nicolas Gallagher 和 Jonathan Neal 出色的 normalize.css（http://necolas.github.com/normalize.css/）。下面的方法并不是他们发明的，他们借用了 https://gist.github.com/413930 里的代码并去掉了注释。上面这个 GitHub 的链接里包含了对这一段 CSS 的详细解释，建议你去看一看。

同时，你可以在自己的项目里使用 normalize.css，推荐你下载这个文件。该文件可以帮你建立一个跨浏览器的统一基准样式表，其文档也很详尽（参见 11.4 节）。

```
/*
* 在所有浏览器中防止sub和sup影响line-height
* gist.github.com/413930
*/
sub,
sup {
    font-size: 75%;
    line-height: 0;
    position: relative;
    vertical-align: baseline;
}
sup {
    top: -0.5em;
}
sub {
    bottom: -0.25em;
}
```

你可能还需要根据内容的字号对这个 CSS 做一些调整，使各行行高保持一致，但上面的代码至少已经为你建立了很好的起点。关于如何创建样式表并将 CSS 添加到网站，参见第 8 章。

4.11 添加作者联系信息

你或许以为 address 元素是用于标记通讯地址的，但其实并非如此（有一种例外的情况，参见第一条提示）。实际上，没有专门用于标记通讯地址的 HTML 元素。

实际上，address 元素是用以定义与 HTML 页面或页面一部分（如一篇报告或新文章）有关的作者、相关人士或组织的联系信息，通常位于页面底部或相关部分内（参见图 4.11.1 和图 4.11.2）。至于 address 具体表示的是哪一种信息，取决于该元素出现的位置。下面的第 (1) 步中对每一种场景都做了描述。

提供作者联系信息

(1) 如果要为一个 article 提供作者联系信息，就将光标放在该元素内（参见图 4.11.1 中第一个例子）。如果要提供整个页面的作者联系信息，就将光标放在 body 中（更常见的做法是放在页面级的 footer 里），参见图 4.11.1 中第二个例子。

(2) 输入 <address>。

(3) 输入作者的电子邮件地址、指向联系信息页的链接等。

(4) 输入 </address>。

提示 大多数时候，联系信息的形式是作者的电子邮件地址或指向联系信息页的链接。联系信息也有可能是作者的通讯地址，这时将地址用 address 标记就是有效的。但是用 address 标记公司网站“联系我们”页面中的办公地点，则是错误的用法。4.16 节中提供了一种标记通讯地址（街道地址）的方法。

```
...
<body>
<main role="main">
<article>
    <h1>Museum Opens on the Waterfront</h1>
    <p>The new art museum not only
    → introduces a range of contemporary
    → works to the city, it's part of
    → larger development effort on the
    → waterfront.</p>

    ... [故事内容的其余部分] ...

    <!-- 文章的页脚，其中包含文章的地址信息 -->
    <footer>
       <p>Tracey Wong has written for
       → <cite>The Paper of Papers</cite>
       → since receiving her MFA in art
       → history three years ago.</p>
       <address>
       Email her at <a href="mailto:
       → traceyw@thepaperofpapers.com">
       →traceyw@thepaperofpapers.com
       → </a>.
       </address>
    </footer>
</article>
</main>

<!-- 页面的页脚，其中包含整个页面的地址信息 -->
<footer role="contentinfo">
    <p><small>&copy; 2014 The Paper of
    → Papers, Inc.</small></p>
    <address>
    Have a question or comment about the
    → site? <a href="site-feedback.html">
    → Contact our web team</a>.
    </address>
</footer>
</body>
</html>
```

图 4.11.1 这个页面有两个 address 元素：一个用于 article 的作者，另一个位于页面级的 footer 里，用于整个页面的维护者。注意 article 的 address 只包含联系信息。尽管 article 的 footer 里也有关于 Tracey Wong 的背景信息，但这些信息是位于 address 元素外面的

4

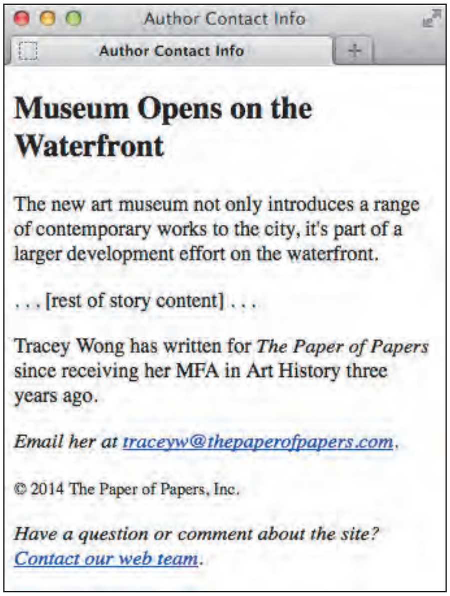


图 4.11.2　address 元素中的文字默认以斜体显示（“The Paper of Papers”也是斜体，不过这是由于它包含在 cite 元素中，参见本章 4.5 节）

提示　如果 address 嵌套在 article 里，则属于其所在的最近的 article 元素；否则属于页面的 body。说明整个页面的作者的联系信息时，通常将 address 放在 footer 元素里，如图 4.11.1 中第二个 address 的实例。

提示　article 里的 address 提供的是该 article 作者的联系信息（图 4.11.1），而不是嵌套在该 article 里的其他任何 article（如用户评论）的作者的联系信息。

提示　address 只能包含作者的联系信息，不能包括其他内容，如文档或文章的最后修改时间（图 4.11.1）。此外，HTML5 禁止在 address 里包含以下元素：h1 ~ h6、article、address、aside、footer、header、hgroup、nav 和 section。

提示　关于 article 的 footer 元素，参见第 3 章。

4.12　标注编辑和不再准确的文本

有时可能需要将在前一个版本之后对页面内容的编辑标出来，或者对不再准确、不再相关的文本进行标记。有两种用于标注编辑的元素：代表添加内容的 ins 元素和标记已删除内容的 del 元素（参见图 4.12.1 至图 4.12.4）。这两个元素既可以单独使用，也可以一起使用。

```
...
<body>

<h1>Charitable Gifts Wishlist</h1>

<p>Please consider donating one or more
→ of the following items to the village's
→ community center:</p>

<ul>
    <li><del>2 desks</del></li>
    <li>1 chalkboard</li>
    <li><del>4 solar-powered tablets
    → </del></li>
    <li><ins>1 bicycle</ins></li>
</ul>

</body>
</html>
```

图 4.12.1　在这个礼品清单上一次发布之后，又增加了一个条目（bicycle），同时根据 del 元素的标注，移除了一些已购买的条目。使用 ins 的时候不一定要使用 del，反之亦然。浏览器通常会让它们看起来与普通文本不一样，参见图 4.12.2

同时，s 元素用以标注不再准确或不再相关的内容（一般不用于标注编辑内容），如图 4.12.5 和图 4.12.6 所示。

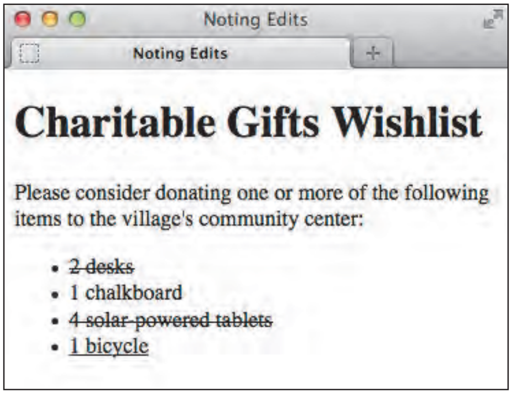


图 4.12.2　浏览器通常对已删除的文本加上删除线，对插入的文本加上下划线。可以用 CSS 修改这些样式

```
...
<body>

<h1>Charitable Gifts Wishlist</h1>
<del>
    <p>Please consider donating one or more
    → of the following items to the village's
    → community center:</p>
</del>

<ins>
    <p>Please note that all gifts have been
    → purchased.</p>
    <p>Thank you <em>so much</em> for your
    → generous donations!</p>
</ins>

<del>
    <ul>
       <li><del>2 desks</del></li>
       <li>1 chalkboard</li>
       <li><del>4 solar-powered tablets
       → </del></li>
       <li><ins>1 bicycle</ins></li>
    </ul>
</del>

</body>
</html>
```

图 4.12.3　del 和 ins 是少有的既可以包围短语内容（HTML5 之前称“行内元素”）又可以包围块级内容的元素，如下面的代码所示



图 4.12.4　与前面类似，浏览器会将删除或插入的内容标出

```
...
<body>

<h1>Today's Showtimes</h1>
<p>Tickets are available for the following
→ times today:</p>

<ol>
    <li><ins>2 p.m. (this show just added!)
    → </ins></li>
    <li><s>5 p.m.</s> SOLD OUT</li>
    <li><s>8:30 p.m.</s> SOLD OUT</li>
</ol>

</body>
</html>
```

图 4.12.5　这个例子展示了一个关于演出时间的有序列表（ol 元素）。与剩余票数不再相关的时段都用 s 元素进行了标记。可以对任何短语使用 s，而不仅限于列表项（li 元素）里的文本。不过，不要像 del 和 ins 那样用 s 标记整个段落或其他块级元素

1. 标记新插入文本

(1) 输入 <ins>。

(2) 输入新内容。

(3) 输入 </ins>。

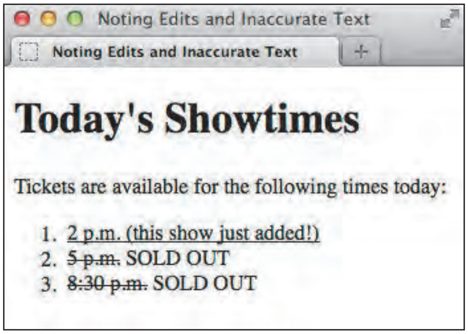


图 4.12.6　默认情况下，浏览器会对 s 元素添加删除线

2. 标记已删除文本

(1) 将光标放在待标记为已删除的文本或元素之前。

(2) 输入 <del>。

(3) 将光标放在待标记为已删除的文本或元素之后。

(4) 输入 </del>。

3. 标记不再准确或不再相关的文本

(1) 将光标放在希望标记为不再准确或不再相关的文本的前面。

(2) 输入 <s>。

(3) 将光标放在希望标记的文本的后面。

(4) 输入 </s>。

提示　del 和 ins 都支持两个属性：cite 和 datetime。cite 属性（区别于 cite 元素）用于提供一个 URL，指向说明编辑原因的页面。例如，<ins cite="http://www.movienews.com/ticket-demand-high.html">2 p.m. (this show just added!)</ins>。datetime 属性提供编辑的时间。（关于 datetime 可接受的格式，参见 4.7 节。）浏览器不会将这两个属性的值显示出来，因此它们的使用并不广泛。不过，应该尽量包含它们，从而为内容提供一些背景信息。它们的值可以通过 JavaScript 或分析页面的程序提取出来。

提示　如果需要向访问者展示内容变化情况，就可以使用 del 和 ins。例如，经常可以看见一些 Web 设计与开发教程使用它们表示初次发布以后新更新的信息，这样可以保持原始信息的完整性。博客、新闻网站等也可以这样做。

提示　用 ins 标记的文本通常会显示一条下划线（图 4.12.2）。由于链接通常也以下划线表示（可能你的网站不是这样，但其他很多网站都是这样），这可能会让访问者感到困惑。可以使用样式表改变插入的段落（或链接）的外观（参见第 10 章）。

提示　用 del 标记的文本通常会显示一条删除线（图 4.12.2）。为什么不直接清除这些文字呢？这取决于你认为是否需要将删除的内容显示出来。加上删除线以后，用户就很容易看出修改了什么。（同时，屏幕阅读器可以读出被删除的内容，不过目前对这一特性的支持还不充分。）

提示　仅在有语义价值的时候使用 del、ins 和 s。如果只是出于装饰的原因要给文字添加下划线或删除线，可以用 CSS 实现这些效果（参见 10.6 节）。

提示　HTML5 指出：“s 元素不适用于指示文档的编辑，要标记文档中一块已移除的文本，应使用 del 元素。”有时，这之间的差异是很微妙的，只能由你决定哪种选择更符合内容的语义。

4.13　标记代码

如果你的内容包含代码示例或文件名，就可以使用 code 元素（参见图 4.13.1 和图 4.13.2）。

```
...
<body>

<p>The <code>showPhoto()</code> function
→ displays the full-size photo of the
→ thumbnail in our <code>&lt;ul id=
→ "thumbnail"&gt;</code> carousel list.</p>

<p>This CSS shorthand example applies a
→ margin to all sides of paragraphs: <code>p
→ { margin: 1.25em; }</code>. Take a look
→ at <code>base.css</code> to see more
→ examples.</p>

</body>
</html>
```

图 4.13.1　code 元素表示其中的文本是代码或文件名。如果你的代码需要显示 < 或 >，应分别使用 < 和 >。这里第二个 code 示例演示了这一点。如果真的用了 < 和 >，浏览器会将这些代码当做 HTML 元素处理，而不是当做文本处理

示例展示了句子中的代码。要显示单独的一块代码（位于句子以外），可以用 pre 元素包住 code 元素以维持其格式（具体的示例参见 4.14 节）。

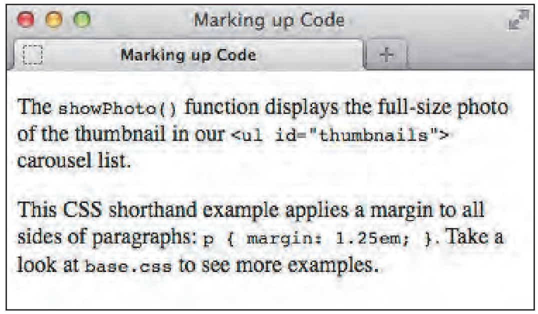


图 4.13.2　code 元素的文本看起来像代码，因为其默认字体为等宽字体

标记代码或文件名的步骤

(1) 输入 <code>。

(2) 输入代码或文件名。

(3) 输入 </code>。

提 示　可以通过 CSS 改变 code 默认的等宽字体样式（图 4.13.2），参见第 10 章。

提 示　关于字符实体（图 4.13.1），参见 1.4 节。

其他计算机相关元素：kbd、samp 和 var

kbd、samp 和 var 元素极少使用，不过你可能会在内容中用到它们。下面对它们作简要说明。

◎ kbd 元素

使用 kbd 标记用户输入指示。

```
<p>To log into the demo:</p>
<ol>
    <li>Type <kbd>tryDemo</kbd> in the User Name field</li>
    <li><kbd>TAB</kbd> to the Password field and type <kbd>demoPass</kbd></li>
    <li>Hit <kbd>RETURN</kbd> or <kbd>ENTER</kbd></li>
</ol>
```

与 code 一样，kbd 默认以等宽字体显示。

◎ samp 元素

samp 元素用于指示程序或系统的示例输出。

```
<p>Once the payment went through, the site returned a message reading,<samp>Thanks
➝ for your order!</samp></p>
```

samp 也默认以等宽字体显示。

◎ var 元素

var 元素表示变量或占位符的值。

```
<p>Einstein is best known for <var>E</var>=<var>m</var><var>c</var><sup>2</sup>.</p>
```

var 也可以作为内容中占位符的值，例如，在填词游戏的答题纸上可以放入 <var>adjective</var>, <var>verb</var>。

var 默认以斜体显示。

需要注意的是，可以在 HTML5 页面中使用 math 等 MathML 元素表示高级的数学相关的标记。更多信息参见 http://dev.w3.org/html5/spec-author-view/mathml.html。

4.14　使用预格式化的文本

通常，浏览器会将所有额外的回车和空格压缩，并根据窗口的大小自动换行。预格式化的文本可以保持文本固有的换行和空格。它是计算机代码示例的理想元素，如图 4.14.1 所示，不过你也可以将它用于文本（比如，ASCII 艺术[①]）。

使用预格式化文本的步骤

(1) 输入 <pre>。

(2) 输入或复制希望以原样显示的文本，包括所需要的空格、回车和换行。除非是代码，不要用任何 HTML（如 p 元素）标记这些文本。

(3) 输入 </pre>。

```
...
<body>

<p>Add this to your style sheet if you want
➝ to display a dotted border underneath the
➝ <code>abbr</code> element whenever it has
➝ a <code>title</code> attribute.</p>

<pre>
    <code>
    abbr[title] {
       border-bottom: 1px dotted #000;
    }
    </code>
</pre>

</body>
</html>
```

图 4.14.1　对于包含重要的空格和换行的文本（如这里显示的 CSS 代码），pre 元素是非常适合的。同时要注意 code 元素的使用，该元素可以标记 pre 外面的代码块或与代码有关的文本（更多细节参见 4.13 节）

① ASCII 艺术指的是用计算机字符（主要是 ASCII 字符）表示图片的一种艺术形式，通常要求使用等宽字体显示。——译者注

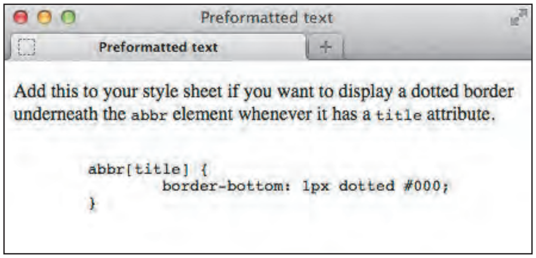


图 4.14.2　注意 pre 内容里的缩进和换行都被保留了

提 示　预格式化的文本通常以等宽字体（如 Courier、Courier New 等）显示，参见图 4.14.2。可以使用 CSS 改变字体样式（参见第 10 章）。

提 示　如果要显示包含 HTML 元素的内容（如教程中的代码示例），应将包围元素名称的 < 和 > 分别改为其对应的字符实体 < 和 >（有关示例参见 4.13 节）。否则，浏览器就会试着显示这些元素。

提 示　一定要对页面进行验证，检查是否在 pre 中嵌套了 HTML 元素（参见 20.1 节）。

提 示　不要将 pre 作为逃避以合适的语义标记内容和用 CSS 控制样式的快捷方式。例如，如果你想发布一篇在字处理软件中写好的文章，不要为了保留原来的格式，简单地将它复制、粘贴到 pre 里。相反，应该使用 p（以及其他相关的文本元素）标记内容，编写 CSS 控制页面的布局。

提 示　同段落一样，pre 默认从新一行开始显示（图 4.14.2）。

---

pre 在表现方面的注意事项

注意，浏览器通常会对 pre 里面的内容关闭自动换行，因此，如果这些内容很宽，就会影响页面的布局，或产生横向滚动条。下面的 CSS 规则可以对 pre 的内容打开自动换行，但在 Internet Explorer 7 及以下版本中并不适用。（随着浏览器市场占有率的改变，在大多数情况下，这些情况都不再需要考虑了。）

```
pre {
    white-space: pre-wrap;
}
```


与之相关的一点提示是，在大多数情况下不推荐对 div 等元素使用 white-space: pre 以代替 pre，因为空格可能对这些内容（尤其是代码）的语义非常重要，而只有 pre 才能始终保留这些空格。（同时，如果用户在其浏览器中关闭了 CSS，格式就丢失了。）

我们将从第 7 章开始讲解 CSS，在第 10 章中讨论文本的格式化。

4.15　突出显示文本

我们都在某些时候用过荧光笔——也许是在复习考试，抑或是在审查合同。无论是哪种情况，荧光笔标记的都是一些与任务有关的关键字词。

HTML5 使用新的 mark 元素实现同样的目的。试着将 mark 想象成荧光笔的语义化对照物。换句话说，重要的是对特定的词语进行标注，与它们如何显示无关。可以用 CSS 对 mark 元素里的文字应用样式（不应用样式也可以），但应仅在合适的情况下使用该元素。

无论何时使用 mark，该元素总是用于提起读者对特定文本片段的注意。下面是一些应用的例子。

- 对搜索结果页面或文章中的搜索词进行突出显示。当我们谈到 mark 时，这是最常见的应用场景。假设使用网站的搜索功能查找“solar panels”。搜索结果或每个结果文章可以使用 <mark>solar panels</mark>，从而在整篇文字中突出显示该词。
- 对于某段引述，如果作者在原来的格式中没有对其进行突出显示，可以用 mark 引起对它的注意（参见图 4.15.1 和图 4.15.2）。这与现实世界中的做法是类似的。
- 引起对代码片段的注意（参见图 4.15.3 和图 4.15.4）。

```
...
<body>

<p>So, I went back and read the instructions
→ myself to see what I'd done wrong. They
→ said:</p>

<blockquote>
    <p>Remove the tray from the box. Pierce
    → the overwrap several times with a
    → fork and cook on High for <mark>15
    → minutes</mark>, rotating it half way
    → through.</p>
</blockquote>

<p>I thought he'd told me <em>fifty</em>. No
→ wonder it exploded in my microwave.</p>

</body>
</html>
```

图 4.15.1　mark 最常见的使用场合是在搜索结果页面，不过，这里给出了另一种合理的用法。包装上的说明中没有突出显示“15 minutes”一词，但这个 HTML 的作者却使用 mark 突出显示了这个词

突出显示文本的步骤

(1) 输入 <mark>。

(2) 输入希望引起注意的字词。

(3) 输入 </mark>。

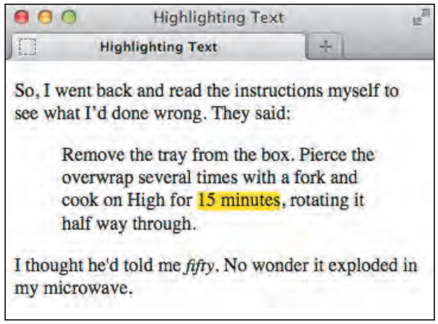


图 4.15.2　对 mark 原生支持的浏览器将对该元素的文字默认加上黄色背景。旧的浏览器不会这样做，但通过一条简单的样式表规则就可以让它们实现这种效果（参见提示）

```
...
<body>

<p>It's usually bad practice to use a class
→ name that explicitly describes how an
→ element should look, such as the
→ highlighted portion of CSS below:</p>

<pre>
    <code>
        <mark>.greenText</mark> {
            color: green;
        }
    </code>
</pre>

</body>
</html>
```

图 4.15.3　这个例子使用 mark 引起对代码片段的注意

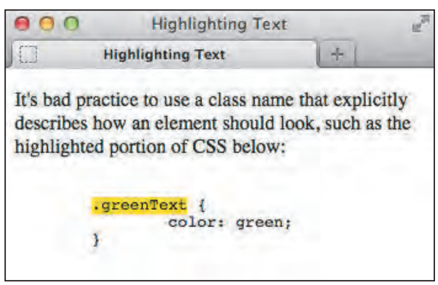


图 4.15.4　浏览器突出显示了由 mark 标记的代码

提示 mark 元素与 em（表示强调）或 strong（表示重要）不同，这两个元素在本章前面讲过了。

提示 由于 mark 是 HTML5 的新元素，因此旧的浏览器不会默认加上黄色背景。可以在样式表中加上 mark { background-color: yellow; } 让这些浏览器实现同样的效果。

提示 不要仅仅因为要给文字加上背景颜色或其他视觉上的考虑而使用 mark。如果只是要给一块文字添加样式，又没有合适的语义化元素，可以使用 span 元素（本章后面将讲到），并用 CSS 为其添加样式（span 元素可能需要添加一个 class）。

4.16 创建换行

浏览器会根据包含内容的块或窗口的宽度让文本自动换行。大多数情况下，让内容像这样充满整行是很合适的，但有时你希望手动地强制文字进行换行。可以使用 br 元素实现这一要求。

要确保使用 br 是最后的选择，因为该元素将表现样式带入了 HTML，而不是让所有的呈现样式都交由 CSS 控制。例如，不要使用 br 模拟段落之间的距离。相反，应该用 p 标记两个段落并通过 CSS 的 margin 属性规定两段之间的距离。

那么，什么时候该用 br 呢？实际上，对于诗歌、街道地址（图 4.16.1 和图 4.16.2）等应该紧挨着出现的短行，都适合用 br 元素。

```
…
<body>

<p>53 North Railway Street<br />
Okotoks, Alberta<br />
Canada T1Q 4H5</p>

<p>53 North Railway Street <br />Okotoks,
→ Alberta <br />Canada T1Q 4H5</p>

</body>
</html>
```

图 4.16.1 同样的地址出现了两次，不过出于演示的目的，对它们的编码有所不同。记住，浏览器总是会忽略代码里的回车，因此两个段落的显示效果是一样的（图 4.16.2）

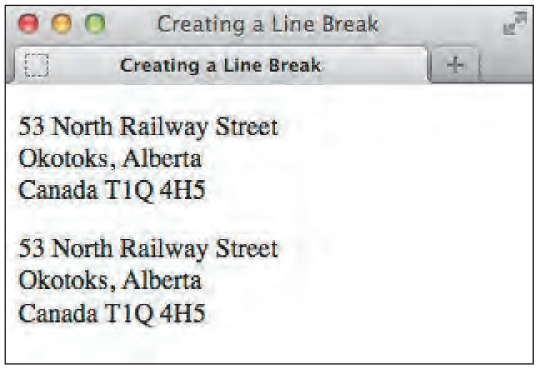


图 4.16.2 每个 br 元素强制让接下来的内容在新的一行显示。如果没有 br 元素，整个地址都会显示在同一行（除非浏览器窗口太窄导致内容换行）

插入换行的方法

在需要换行的地方输入
（或
）。没有单独的 br 结束标签，因为它是所谓的**空元素**，没有任何内容。

提示 在 HTML5 中，输入
 或
 都是有效的。

提示　可以使用 CSS 控制段落中的行间距（参见 10.7 节）以及段落之间的距离（参见 11.8 节）。

提示　hCard 微格式（http://microformats.org/wiki/hcard）是用于表示人、公司、组织和地点的人类和机器都可读的语义形式。可以使用微格式替代图 4.16.1 中表示街道地址的方式。

4.17 创建 span

同 div 一样，span 元素是没有任何语义的。不同的是，span 只适合包围字词或短语内容，而 div 适合包含块级内容（参见 3.12 节）。

如果你想将下面列出的项目应用到某一小块内容，而 HTML 又没有提供合适的语义化元素，就可以使用 span。

- 属性，如 class、dir、id、lang、title 等（图 4.17.1 和图 4.17.2）。
- CSS 样式。
- JavaScript 行为。

由于 span 没有任何语义，因此应将它作为最后的选择，仅在没有其他合适的元素时才使用它。

```
...
<body>

<h1 lang="es">La Casa Milà</h1>

<p>Gaudí's work was essentially useful.
→ <span lang="es">La Casa Milà</span> is
→ an apartment building and <em>real people
→ </em> live there.</p>

</body>
</html>
```

图 4.17.1　在这个例子中，我想对一小块文字指定不同的语言，但从句子的上下文看，没有一个语义上适合“La Casa Milà”的 HTML 元素。段落之前包含“La Casa Milà”的 h1 在语义上是合适的，因为这些文字就是后面内容的标题。因此对标题来说，直接在 h1 中添加 lang 属性就可以了，不必为了指定语言而将标题包在一个 span 中（lang 属性用于对元素的文本指定语言）



图 4.17.2　span 元素没有任何默认样式

添加 span 的步骤

(1) 输入 <span。

(2) 如果需要，输入 id="*name*"，其中 *name* 用于唯一地标识 span 包含的内容。

(3) 如果需要，输入 class="*name*"，其中 *name* 是 span 所属类的名称。

(4) 如果需要，输入其他的属性（如 dir、lang、title 等）及其值。

(5) 输入 > 结束开始标签。

(6) 创建希望包含在 span 里的内容。

(7) 输入 。

提示　span 没有任何默认格式（参见图 4.17.2），但就像其他 HTML 元素一样，可以用 CSS 添加你自己的样式。

提示　可以对一个 span 元素同时添加 class 和 id 属性，但通常只应用这两个中的一个（如果真要添加的话）。主要区别在于，class 用于一组元素，而 id 用于标识页面中单独的、唯一的元素。

提 示　在 HTML 没有提供合适的语义化元素时，微格式经常使用 span 为内容添加语义化类名，以填补语义上的空白。要了解更多信息，参见 http://microformats.org。

4.18　其他元素

本节讲的是 HTML 文本中可能用到的其他元素，但它们通常只在极少数情况下才会用到，或者浏览器对它们的支持还不完善（或兼而有之）。

1. u 元素

同 b、i、s 和 small 一样，HTML5 重新定义了 u 元素，使之不再是无语义的、用于表现的元素。以前，u 元素用来为文本添加下划线。现在，u 元素用于非文本注解（听起来确实有些迷惑）。HTML5 对它的定义为：

> u 元素为一块文字添加明显的非文本注解，比如在中文中将文本标为专有名词（即中文的专名号[①]），或者标明文本拼写有误。

下面是使用 u 标注拼错的词的例子：

```
<p>When they <u class="spelling">
→ recieved</u> the package, they put
→ it with <u class="spelling">there
→ </u> other ones with the intention
→ of opening them all later.</p>
```

class 完全是可选的，它的值（可以是你想填的任何内容）也不会在内容中明显指出这是个拼写错误。不过，可以用它对拼错的词添加不同于普通文本的样式（u 默认仍以下划线显示）。通过 title 属性可以为该元素包含的内容添加注释，如“[sic]”（在一些语言中用于指示拼写错误的惯用符号）。

仅在 cite、em、mark 等其他元素语义上不合适的情况下使用 u 元素。同时，最好改变 u 文本的样式，以免与同样默认添加下划线的链接文本弄混（图 4.18.1）。

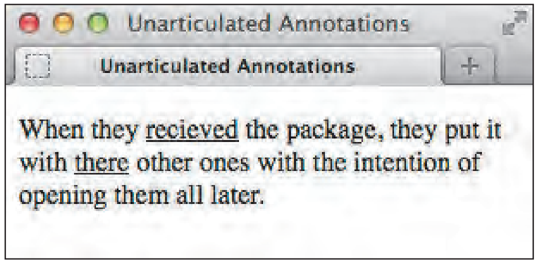


图 4.18.1　同链接一样，浏览器默认为 u 元素添加下划线。除非用 CSS 改变它们的样式，否则容易引起混淆

2. wbr 元素

HTML5 为 br 引入了一个相近的元素，称为 wbr。它代表“一个可换行处”。可以在一个较长的无间断短语（如 URL）中使用该元素，表示此处可以在必要的时候进行换行，从而让文本在有限的空间内更具可读性。因此，与 br 不同，wbr 不会强制换行，而是让浏览器知道哪里可以根据需要进行换行。

下面是一些例子：

```
<p>They liked to say, "FriendlyFleas
→ andFireFlies<wbr /> FriendlyFleasa
→ ndFireFlies<wbr />FriendlyFleasand
→ FireFlies<wbr />" as fast as they
→ could over and over.</p>
<p>His favorite site is this<wbr />
→ is<wbr />a<wbr />really<wbr />
→ really<wbr />longurl.com.</p>
```

输入 wbr 时，既可以用 <wbr />，也可以用 <wbr>。或许你已经猜到了，用到 wbr 的场合并不多。而且，截至本书写作之际，浏览器对它的支持并不一致。尽管在当前版

① 专名号用于表示人名、地名、朝代名等专名。——译者注

本的 Chrome 和 Firefox 中 wbr 是有效的，但 Internet Explorer 和 Opera 会忽略它。

3. ruby、rp 和 rt 元素

旁注标记（ruby annotation）是东亚语言（如中文和日文）中一种惯用符号，通常用于表示生僻字的发音。这些小的注解字符出现在它们标注的字符的上方或右方。它们常简称为**旁注**（ruby 或 rubi）。日语中的旁注字符称为**振假名**（furigana）。

ruby 元素以及它们的子元素 rt 和 rp 是 HTML5 中为内容添加旁注标记的机制。rt 指明对基准字符进行注解的旁注字符。可选的 rp 元素用于在不支持 ruby 的浏览器中的旁注文本周围显示括号。

下面的例子用英文占位符展示了这种结构，帮助你理解代码中以及支持和不支持它们的浏览器中对各类信息的安排方式（支持的浏览器效果如图 4.18.2，不支持的效果如图 4.18.3）。浏览器将旁注文本进行了突出显示：

```
<ruby>
    base <rp>(</rp><rt>ruby chars
    → </rt><rp>)</rp>
    base <rp>(</rp><rt>ruby chars
    → </rt><rp>)</rp>
</ruby>
```

现在，展示一个真实例子。例子中有两个表示“Beijing”的中文基准字符，以及伴随它们的旁注字符（图 4.18.4）：

```
<ruby>
    北 <rp>(</rp><rt>ㄅㄟˇ</rt><rp>)</rp>
    京 <rp>(</rp><rt>ㄐㄧㄥ</rt><rp>)</rp>
</ruby>
```

可以看到在不支持 ruby 的浏览器中括号的重要性（参见图 4.18.5）。没有它们，基准字符和旁注文本就会显示在一起，让内容变得混乱。

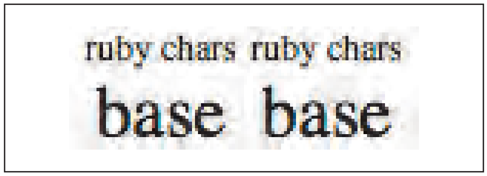


图 4.18.2　支持旁注标记的浏览器会将旁注文本显示在基准字符的上方（也可能在旁边），不显示括号

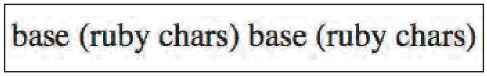


图 4.18.3　不支持旁注标记的浏览器会将旁注文本显示在括号里，就像普通的文本一样



图 4.18.4　支持旁注的浏览器中显示的“Beijing”的旁注标记

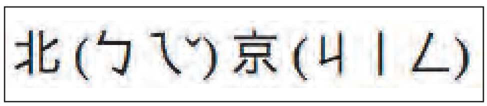


图 4.18.5　在不支持旁注的浏览器中，如果没有括号，内容就会变得难以理解

提 示　在本书写作之际，Firefox 和 Opera 还缺少基本的 ruby 支持（这也是使用 rp 的原因）。不过，Firefox 插件 HTML Ruby（https://addons.mozilla.org/en-US/firefox/addon/html-ruby/）为 Firefox 提供了对旁注的支持。

提 示　关于旁注字符的更多信息，参见 http://en.wikipedia.org/wiki/Ruby_character。

4. bdi 和 bdo 元素

如果你的 HTML 页面中混合了从左至右书写的字符（如大多数语言所用的拉丁字符）和从右至左书写的字符（如阿拉伯语或希伯来语字符），就可能要用到 bdi 和 bdo 元素。

不过，首先要讲一点背景知识。除非在 html 元素中添加 dir 属性并将属性值设为 rtl，否则内容的基准方向都默认为从左至右。例如，<html **dir="rtl"** lang="he"> 指明内容的方向为从右至左，且基准语言为希伯来语。

就像我在全书多个示例中对 lang 的处理方式一样，对于页面内的元素，如果其内容与页面基准设置不一致，也可以对其设置 dir 属性。因此，如果基准语言设为英语（<html lang="en">），又要包含一段希伯来语，就可以标记为 <p dir="rtl" lang="he">...</p>。

单独进行了设置的内容一般将按照希望的方向显示。这是由 Unicode 的双向（bidi）算法控制的。

如果上述算法**不**按设想的方式显示文本，可以使用 bdo（bidirectional override，双向重载）元素对内容进行重载。通常，这代表 HTML 源代码中的内容是视觉顺序而非逻辑顺序的情况。

视觉顺序（visual order）就是内容看上去的顺序——HTML 源代码内容与你希望显示的顺序相同。**逻辑顺序**（logical order）则与之相反，可用于希伯来语这样从右至左书写的语言。先输入从右至左的第一个字符，然后是第二个（也就是第一个字符左边的字符），以此类推。

根据最佳实践，Unicode 希望双向文本都以逻辑顺序呈现。因此，如果文本是以视觉顺序呈现的，算法仍会让字符反向，显示出与预期相反的顺序。如果你无法将 HTML 源代码中的文本转换为逻辑顺序（例如，这些文本可能来自数据库或源），就只能依靠 bdo 了。

要使用 bdo，必须包含 dir 属性并将属性值设为 ltr（由左至右）或 rtl（由右至左），指定你希望呈现的方向。继续先前在英文页面中包含希伯来语段落的例子，应输入 <p lang="he">**<bdo dir="rtl">...**</bdo></p>。bdo 适用于段落里的短语或句子。不能用它包围多个段落。

bdi 元素是 HTML5 中新加的元素，用于内容的方向未知的情况。不必包含 dir 属性，因为默认已设为自动判断。下面的例子来自 HTML5 规范（但有少许修改）：

> 这个元素特别适用于包围用户生成的方向未知的内容。

在这个例子中，用户名与用户提交的帖子的数量显示在一起。如果不使用 bdi 元素，阿拉伯用户的用户名将让文本变得难以理解（双向算法会把冒号和数字“3”置于“User”一词旁边，而不是在“posts”一词旁边）。

```
<ul>
    <li>User <bdi>jcranmer</bdi>:
    ➝ 12 posts.</li>
    <li>User <bdi>hober</bdi>:
    ➝ 5 posts.</li>
    <li>User <bdi>إيان</bdi>:
    ➝ 3 posts.</li>
</ul>
```

提示 要了解关于由右至左语言的更多信息，推荐阅读 W3C 的文章“Creating HTML Pages in Arabic, Hebrew, and Other Right-to-Left Scripts”（www.w3.org/International/tutorials/bidi-xhtml/）。

---

5. meter 元素

meter 元素也是 HTML5 的新元素。乍看起来，它很像 progress 元素（接下来会讲到，

根据规范，它表示“任务的完成进度”）。

可以用 meter 元素表示分数的值或已知范围的测量结果。简单地说，它代表的是投票结果（如“30% Smith，37% Garcia，33% Hawkins”）、已售票数（如“共 850 张，已售 811 张”）、考试分数（如“百分制的 90 分”）、磁盘使用量（如“256 GB 中的 74 GB”）等测量数据。

HTML5 建议（并非强制）浏览器在呈现 meter 时，在旁边显示一个类似温度计的图形——一个表示测量值的横条，测量值的颜色与最大值的颜色有所区别（相等除外）。作为当前少数几个支持 meter 的浏览器，Firefox 正是这样显示的（参见图 4.18.6）。对于不支持 meter 的浏览器，可以通过 CSS 对 meter 添加一些额外的样式，或用 JavaScript 进行改进。

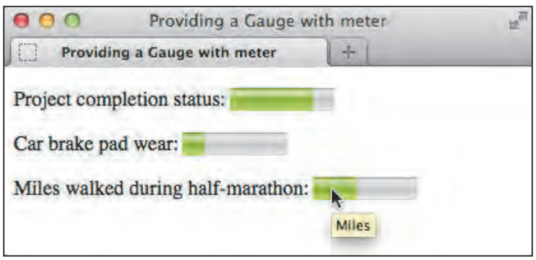


图 4.18.6　支持 meter 的浏览器（如 Firefox）会自动显示测量值，并根据属性值进行着色。<meter> 和 </meter> 之间的文字不会显示出来。如最后一个例子所示，如果包含 title 文本，就会在鼠标悬停在横条上时显示出来

虽然并非必需，但最好在 meter 里包含一些反映当前测量值的文本，供不支持 meter 的浏览器显示，如图 4.18.7 所示。

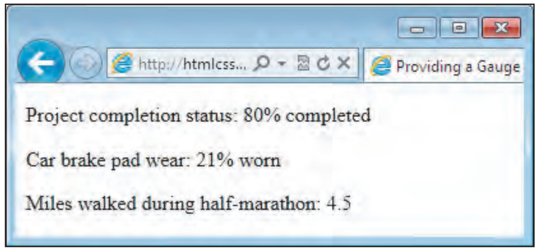


图 4.18.7　IE9 不支持 meter，它会将 meter 元素里的文本内容显示出来，而不是显示一个彩色的横条。可以通过 CSS 改变其外观

下面是一些 meter 的例子（显示结果如图 4.18.6 和图 4.18.7 所示）：

```
<p>Project completion status: <meter
→ value="0.80">80% completed</meter>
→ </p>
<p>Car brake pad wear: <meter low=
→ "0.25" high="0.75" optimum="0"
→ value="0.21">21% worn</meter></p>
<p>Miles walked during half-marathon:
→ <meter min="0" max="13.1" value="5.5"
→ title="Miles">4.5</meter></p>
```

meter 不提供定义好的单位，但可以使用 title 属性指定单位，如最后一个例子所示。通常，浏览器会以提示框的形式显示 title 文本（图 4.18.7）。

提 示　meter 支持好几个属性。value 是其中唯一必需包含的属性。如果不指定 min（最小值）和 max（最大值），则默认将它们分别设为 0 和 1.0。low、high 和 optimum 属性通常共同作用，它们将范围划分为低、中、高三个区间。optimum 代表范围内的最优位置，如上文例子中的“0 brake pad wear”。如果 low 和 high 的值均不是最优的，可以将 optimum 设为它们之间的值。

提示 截至本书写作之际，浏览器对 meter 的支持情况还在变化：Internet Explorer、移动 Safari（用于 iOS 设备）和 Android 浏览器还不支持该元素。这大概也是现实中很少见到它的原因。随意使用它，只是需要了解大多数浏览器默认只显示 meter 文本而不显示图形（图 4.18.7）。关于最新的浏览器支持情况，参见 http://caniuse.com/#feat=progressmeter。

提示 支持 meter 的不同浏览器显示测量值图形的样式可能有差异。

提示 已经有人试过针对支持 meter 的浏览器和不支持的浏览器统一编写 meter 的 CSS。在网上搜索“style HTML5 meter with CSS”（用 CSS 为 HTML5 的 meter 添加样式），就可以找到一些解决方案（注意其中的一些用到了 JavaScript）。

提示 meter 并不用于标记没有范围的普通测量值，如高度、宽度、距离、周长等。例如，这种写法是不正确的：`<p>I walked <meter value="4.5">4.5</meter> miles yesterday.</p>`。

提示 一定不要将 meter 和 progress 元素混在一起使用。

6. progress 元素

progress 元素也是 HTML5 的新元素。前面说过，它指示某项任务的完成进度。可以用它表示一个进度条，就像在 Web 应用中看到的指示保存或加载大量数据操作进度的那种组件。

就像 meter 一样，支持 progress 的浏览器会根据属性值自动显示一个进度条（参见图 4.18.8）。同时，和 meter 一样，为了让旧的浏览器也能表现进度，尽管并非必需，但最好在 progress 中包含反映当前进度的文本（如示例中的“0% saved”，参见图 4.18.9）。

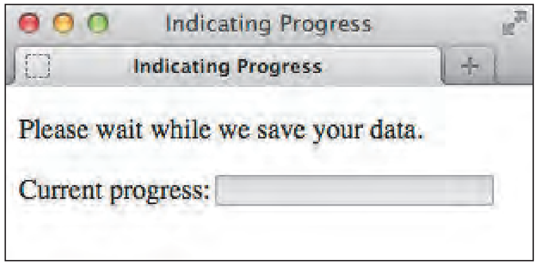


图 4.18.8 支持 progress 的浏览器（如 Firefox）会自动显示进度条，并根据值对其进行着色。<progress> 和 </progress> 之间的文本不会显示出来。在这个例子中 value 属性设成了 0，因此整个横条都是同样的颜色

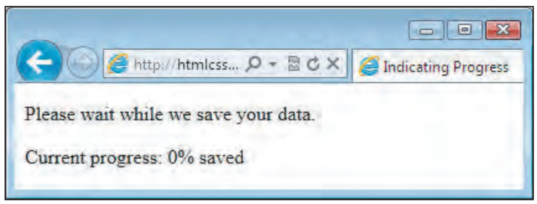


图 4.18.9 IE9 不支持 progress，因此不会显示有颜色的横条，而是显示该元素里面的文本。可以通过 CSS 改变其外观

下面是一个例子：

```
<p>Please wait while we save your
→ data.</p>

<p>Current progress: <progress
→ max="100" value="0">0% saved
→ </progress></p>
```

对 progress 的完整说明已经超出了本书的范围。通常，你只能通过 JavaScript 动态地更新 value 属性值和元素里面的文本以指示任务进程（例如，指示已完成 37%）。使用 JavaScript 与在 HTML 中硬编码从视觉上看是

一样的，即

```
<progress max="100" value="37">37%
saved</progress>
```

效果如图 4.18.10 所示。当然，不支持的浏览器会显示为类似于图 4.18.9 所示的样子。

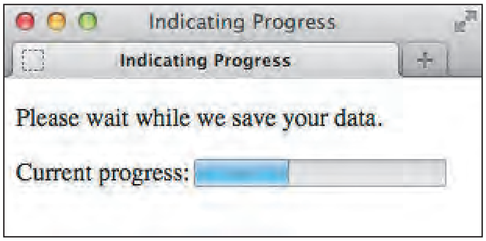


图 4.18.10　Firefox 中显示的进度条，通过 JavaScript（或直接在 HTML 中）将 value 属性设为 37（假定 max="100"）

提示　progress 元素支持三个属性：max、value 和 form。它们都是可选的，max 属性指定任务的总工作量，其值必须大于 0。value 是任务已完成的量。如果 progress 没有嵌套在 form 元素里面，又需要将它们联系起来，可以添加 form 属性并将其值设为该 form 的 id。

提示　下面简单地看看如何用 JavaScript 修改 progress 元素。假定进度条已经设定了一个 id，如：

```
<progress max="100" value="0" id=
→ "progressBar">0% saved</progress>
```

下面的 JavaScript 可以让你获取该元素：

```
var bar = document.getElementById
→ ('progressBar');
```

然后就可以通过 bar.value 获取或设定 value 的值。例如，**bar.value=37;** 可以将值设为 37。

提示　在本书写作之际，大多数桌面浏览器的最新版本都支持 progress 元素。IE9 及之前的版本、移动 Safari 和 Android 浏览器还不支持该元素。关于最新的浏览器支持情况，参见 http://caniuse.com/#feat=progressmeter。

提示　不同浏览器显示进度条的样式可能不同，不过可以通过 CSS 为它添加样式。

第 5 章

图像

5

本章内容

- ❑ 关于 Web 图像
- ❑ 获取图像
- ❑ 选择图像编辑器
- ❑ 保存图像
- ❑ 在页面中插入图像
- ❑ 提供替代文本
- ❑ 指定图像尺寸
- ❑ 在浏览器中改变图像的尺寸
- ❑ 在图像编辑器中改变图像的尺寸
- ❑ 为网站添加图标

为 Web 创建图像与为在纸上输出而创建图像有所不同。尽管 Web 图像和可打印图像的基本性质是相同的，但它们在格式、下载速度、颜色、大小（尺寸）、透明度和动画等六个主要方面有一些区别。

本章将阐释这六个要素的重要方面，以及如何使用这些知识为网站创建有效的图像。另外，还将学习如何在网页中插入图像。

5.1 关于 Web 图像

让我们来看看创建 Web 图像时应记住的六个要素。如果你不愿意花时间了解具体的细节，可以直接跳至节末的“小结”。

1. 格式与下载速度

在纸上打印图像的人不必担心读者将使用什么查看图像，读者在打开杂志或报纸的页面时也不必等待图像的出现。不过，在万维网上，情况就不一样了。

每天，人们通过数以万计的 Mac、基于 Windows 的 PC、Linux 机器、手机、平板电脑等各种各样的设备访问万维网。网页中的图像应该采用这些设备的操作系统都能识别的格式。当前，Web 上用的最广泛的三种格式是 GIF、PNG 和 JPEG。我们的目标是选择质量最高，同时文件最小的格式。

◎ JPEG

JPEG 格式适用于彩色照片，因为它包含大量的颜色并进行了合理的压缩，采用这种格式保存的文件相对较小（参见图 5.1.1）。图像的文件变小（且不论是何种格式），下载速度就会变快，访问者就不必等待较长时间才能看到图像。



图 5.1.1　全彩的照片通常保存为 JPEG 格式。PNG-24 格式效果也很好，但通常其文件大小比 JPEG 要大得多

不过，JPEG 是一种**有损的**格式，因此在将图像保存为 JPEG 时会丢失一部分原始信息，但通常有必要这样做，因为我们可以将图像质量的损失控制在用户不易觉察的范围内，却能显著改善图像的加载速度。

注意，对 JPEG 进行解压缩无法还原图像先前丢失的细节。因此，如果未来还有可能对图像进行编辑，就应该保存一份使用无压缩格式（如 PSD 或 TIFF）的图像副本，只有在确保不再对图像进行修改的情况下才能只保存 JPEG 格式的图像。

◎ PNG 和 GIF

PNG 和 GIF 是**无损的**格式，因此采用这两种格式对图像进行压缩时不会造成品质的损失。GIF 只有 256 种颜色，但 PNG 却支持几百万种颜色。与 JPEG 不同，PNG 和 GIF 均支持透明，它们更适用于保存非照片类的图像。通常，拥有大片纯色的图像，如标识、重复的图案、插图以及图像文字等都适合使用这两种格式。

可以使用 PNG 保存照片，但由于无损图像质量，文件尺寸会比 JPEG 大得多。因此，只有在压缩造成的质量损失不可忽略的情况下才使用 PNG 保存照片。

PNG 有几种不同的分支：PNG-8、PNG-24 和 PNG-32。一般来说，对于 PNG 和 GIF，应优先选择 PNG，因为它对透明度的支持更好，压缩算法也更好，产生的文件更小。

通过表 5.1.1 可以对比不同的图像格式（包括 PNG 的不同分支）。5.4 节包含了对图像格式和压缩情况进行设置的例子。

表 5.1.1　图像格式对比

格式	用　　法	颜　　色	索引色（基本）透明	alpha 透明
JPEG	适用于大多数照片，以及其他颜色较多且可接受一些质量损失的图像	1600 万以上	—	—
PNG-8	适用于标识、重复的图案以及其他颜色较少的图像或具有连续颜色的图像	256	支持	支持
PNG-24	与 PNG-8 相似，不过支持颜色更多的图像。适用于颜色丰富且质量要求高的照片	1600 万以上	支持	—
PNG-32	与 PNG-24 相似，不过支持具有 alpha 透明的图像	1600 万以上	—	支持
GIF	用法与 PNG-8 相似，在大多数情况下应使用 PNG-8	256	支持	—

WebP 图像格式

谷歌建立了另一种图像格式，名为 WebP。这种格式既支持有损压缩也支持无损压缩，它产生的文件大小也远小于 JPEG 和 PNG。跟 PNG 一样，它还支持 alpha 透明。

WebP 还在发展之中，截至本书写作之际，完全支持这种格式的浏览器还仅限于 Chrome、Opera 12+ 和一些 Android 浏览器，至于其他浏览器是否会支持这种格式尚不可知。不过，有必要关注一下这种格式的进展情况。在很长一段时间内，还没有办法做到让支持它的浏览器显示这种格式的图像，而其他浏览器使用备用格式的图像。要了解 WebP 的详细信息，参见 https://developers.google.com/speed/webp/。最新的浏览器支持情况参见 http://caniuse.com/#search=webp。

2. 颜色

大多数计算机显示器可以显示数以百万计的颜色，但也有例外的情况。有的图像格式的调色板是有限的。GIF 和 PNG-8 图像只有 256 种颜色，对标志和图标来说通常这已经足够了（参见图 5.1.2）。



图 5.1.2　标识和其他颜色较少的图像通常保存为 PNG-8 格式（也可以保存为 GIF 格式，不过相较之下 PNG 更好一些）。另见彩插

JPEG、PNG-24 和 PNG-32 均支持超过 1600 万种的颜色，因此照片和复杂的插图应选择这些格式。不过，需要指出的是，对于这些图像，大多数情况下应使用 JPEG。

3. 大小（尺寸）

你曾经收到过一些在屏幕上看起来特别大的照片吗？这样的照片通常是由数码相机拍摄的，而发送照片给你的人又没有事先编辑文件以缩减尺寸。那么为什么这些照片起初这么大呢？如果将它们放到网站上，应该将文件大小控制在多少呢？

数字图像以像素为单位进行度量。如今，超过 800 万像素的数码相机已经很常见了，不过我们不妨以 300 万像素的数码相机举例说明。300 万像素的数码相机可以照出 2048 像素宽、1536 像素高的照片。这在浏览器里有多大呢？视情况而定。不过一般而言，这样的尺寸**太大了**（参见图 5.1.3），你在邮件中查看这么大的图像就是这个样子。

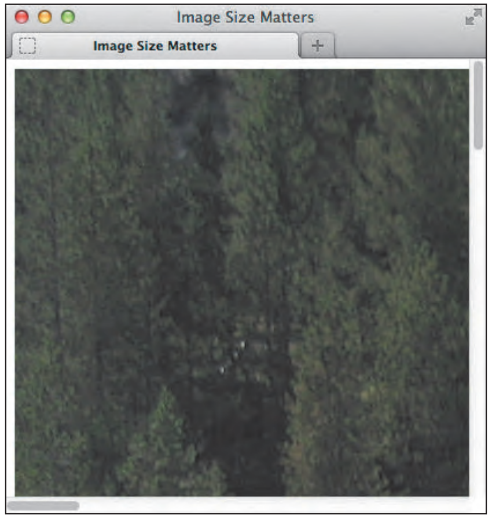


图 5.1.3　这个图像为 2048 像素宽，1536 像素高，这意味着要看到照片的其他部分，既需要纵向滚动条，又需要横向滚动条，更别提文件尺寸了，需要很长时间才能将图像显示完整。这可谓只见树木、不见森林啊（真是令人崩溃）

那么，对于网站上的图像，多大合适呢？简单地说，通常应控制在几百个像素宽（参见图 5.1.4）。图像尺寸越大，文件就越大，网页加载它的时间就越长。选择图像尺寸是一个取舍的过程。此外，你已看到，如果图像尺寸过大，访问者就需要使用横向滚动条来查看（参见图 5.1.3）。

当然，图像的尺寸也取决于其用途。图标通常很小，标识稍大一些，照片则大得多。有时，为了产生视觉冲击，我们可能需要使用一张达到或超过整个网站内容宽度的大图。这样的图像通常不超过 960 像素宽。

打印出来的图像通常比显示器上看到的要小一些。打印机的**每英寸点数**（dpi）通常比显示器的**每英寸像素数**（ppi）要多。屏幕分辨率变大加剧了这一情形。这就是为什么相同的图像在显示器或笔记本上看要比在纸上看要大得多。

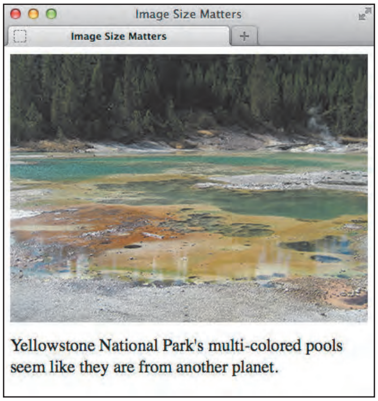


图 5.1.4 看，原来这张图还有这么多内容！我减小尺寸（参见 5.9 节）后很容易就能看到整个图像。这个版本有 400 像素宽，300 像素高，维持了原图 4 ∶ 3 的宽高比，保存了图像的基本形状。我们在第 12 章会了解到，可以使用 CSS 控制图像的宽度和高度，让图像放大或缩小，使之在所有的屏幕（从手机到大的显示器）上都能以最好的效果显示

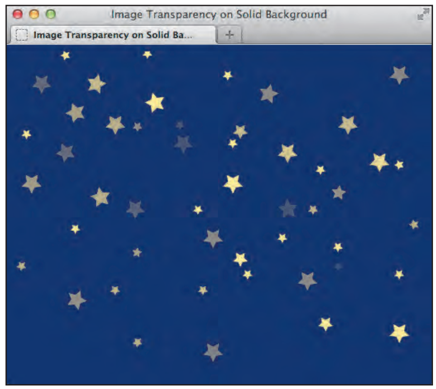
可缩放矢量图形（SVG）

对于使用 SVG 图像语言创建的图像，无论放大还是缩小都不会影响其质量（以及其他的一些参数）。而且，对于某个 SVG 图像来说，无论它在页面中显示的尺寸是多大，其文件大小总是恒定的。目前，几乎所有的现代浏览器都提供基本的 SVG 支持，因此你可以在网页中使用 SVG。不过，IE8 还不支持 SVG。可以使用 SVG Web（http://code.google.com/p/svgweb/）或 Raphaël（http://raphaeljs.com/）等 JavaScript 库在 IE8 中实现类似的效果。更多关于 SVG 的信息（尤其是结合使用 HTML 视频和 SVG），参见 17.14 节。

4. 透明度

可以利用透明度为图像创建非矩形的边缘，在图像的下面设置背景色或图案。PNG 和 GIF 都支持透明度，JPEG 则不支持。

在 GIF 格式中，一个像素要么是透明的，要么是不透明的。这称作**索引色透明**（index transparency）。而 PNG 则既支持索引色透明，又支持 **alpha 透明**（alpha transparency）。alpha 透明可以控制一个像素透明的程度。也就是说，一个像素可以部分透明，而非要么透明要么不透明。这意味着具有复杂透明背景的图像使用 PNG 的效果（参见图 5.1.5）要好于使用 GIF 的效果，因为使用 PNG 可以让边缘变得平滑，避免产生锯齿。



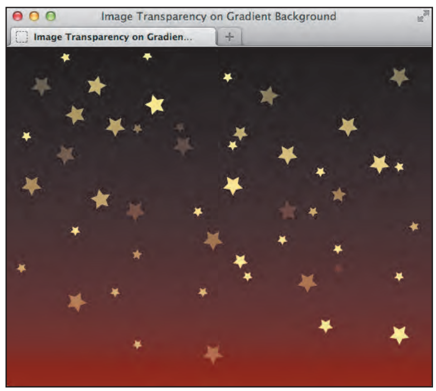


图 5.1.5 纯蓝色背景和从黑到红渐变色背景都不是星星 PNG 图的一部分，它们只是用 CSS 向页面的 body 添加的颜色。星星图具有 alpha 透明度，这样，无论背景是纯色、渐变色，还是另一张图像，就都能穿透，且看起来相当“干净”。JPEG 无法实现这一点，GIF 仅支持一种简单的透明度，效果并不太好（除非背景是纯色的）。另见彩插

PNG-8 既支持索引色透明，也支持 alpha 透明，但需要使用 Fireworks 这样的程序才能将图像保存为 PNG-8 格式。Photoshop 不支持 alpha 透明的 PNG-8，但支持 alpha 透明的 PNG-32。（参见补充材料“Photoshop、PNG-24 与 PNG-32”。）这也是万维网上大多数透明 PNG 都是 PNG-32 的原因。

总之，对于透明图像，应使用 PNG-8 或 PNG-32。后者允许使用超过 256 种颜色。

> **Photoshop、PNG-24 与 PNG-32**
>
> PNG-24 与 PNG-32 几乎是一样的，只是后者支持 alpha 透明。奇怪的是，Photoshop 将 PNG-24 和 PNG-32 都称为 PNG-24，因此通常容易认为 PNG-24 是支持 alpha 透明的。实际上，在 Photoshop 里，如果为 PNG-24 选中透明度选项，它会自动创建 PNG-32 图像。
>
> 其实，大多数人都不清楚这一点。很多人甚至没有听说过 PNG-32。因此，“PNG-24”也常常用于指代带有 alpha 透明的图像，不过，从技术上看这种说法并不准确。

5. 动画

动画可以保存为 GIF，但不能是 JPEG 或 PNG。实际上，使用图像表现动画已经用得越来越少了。（一种例外的情况是那些好玩的 GIF 动画，它们在 Tumblr 这类网站相当流行。）我们通常使用 CSS 动画、JavaScript、HTML5 Canvas、SVG（可谓是一匹黑马）和 Flash 创建动画。近几年，使用 Flash 创建的动画越来越少了。这主要是由于 iOS 不支持 Flash，且其他标准的 Web 技术的能力和浏览器支持程度都提升了很多。

6. 小结

让我们来回顾一下有关 Web 图像的要点。

- 大多数照片都应保存为 JPEG 格式，图标、标识等颜色较少的图像应保存为 PNG 格式。
- 应使用 PNG-8 或 PNG-32（常误作 PNG-24）创建具有 alpha 透明度的图像。
- 合理的图像大小（尺寸）会让图像文件变得更小。应尽可能地减小图像尺寸，从而让页面加载得更快。

5.2　获取图像

如何获取在网页中使用的图像呢？有几种方式。可以购买或下载现成的图像，通过扫描仪将照片或手绘图像数字化，使用数码相机拍摄照片，或使用 Adobe Photoshop 这样的图像编辑软件从头绘制图像。在有了图像以后，就可以将它们保存为上文提到的某种格式并在万维网上使用它们了。

获取图像的手段

- 可以使用搜索引擎寻找万维网上的图像，方法是点击搜索框上方的“图像”（Images）链接，再跟平常一样输入搜索词。注意，一般来说，对于在万维网上找到的图像，即便是免费的，也会受到某种形式的版权限制（参见后面的“知识共享许可协议”）。购买的图像通常可以随意使用，但一般不能对图像本身再次进行销售。请仔细阅读声明或许可协议。
- 很多公司以较低的价格销售库存的照片和图像。通常，每张图像都有几个版本，以满足不同的用途和分辨率要求。
- 数码相机（包括智能手机的相机）可能是创建个人图像最为常用的方法。

5

知识共享许可协议

知识共享（Creative Commons，www.creativecommons.org）是一个非营利组织，它开发了一个版权模板体系，让艺术家可以按照其指定的方式分享他们的作品，同时无需放弃对作品的所有权利。网站设计师、音乐家和摄影师使用知识共享许可协议将他们的作品放入市场，而无需担心别人以他们不允许的方式利用这些作品。

Flickr 是流行的图片分享 Web 应用（www.flickr.com）。它要求用户为他们上传的每张图片指定一种知识共享许可协议，然后访问者可以根据许可协议的类型搜索图片。它是为网站寻找图片的好地方。

此外，使用 Google 搜索引擎可以根据使用权限对搜索进行限定（进入 www.google.com/advanced_search，再在底部“使用权限”（Usage Rights）下拉菜单中选择需要的选项）。

5.3　选择图像编辑器

有很多不同的软件可以用来创建和保存 Web 图像。大多数现代图像编辑器都提供了创建 Web 图像的专用工具。这些工具考虑了本章前面讨论的几种要素。

毫无疑问，行业标准是 Adobe Photoshop（www.adobe.com）。跟它功能相近的 Adobe Fireworks 也非常强大。它们都既有 OS X 版也有 Windows 版。在 2013 年春，Adobe 宣称尽管它们还将继续销售 Fireworks，但不会再添加新的功能。这是在购买该软件时需要考虑的一点。还有很多软件可以作为 Photoshop 和 Fireworks 的替代品（通常也要便宜得多），参见补充材料。

对于为网站准备的图像，这些软件都可以对其进行最基本的处理，如缩放、剪裁、调整颜色、应用效果、优化等。对于收费软件，通常都有免费的试用版。OS X 自带一个非常基础的图像编辑器，名为 Preview。还可以在网上搜索“image editors”（图像编辑器），找到更多的选择。

不过，需要强调一点，无论使用哪种软件，优化 Web 图像的基本策略始终不会改变。它们所用的命令名称可能有所不同，步骤也有多有少，但思路是一样的。

Photoshop 和 Fireworks 的替代品

创建图像并不仅限于 Photoshop 和 Fireworks。以下列举了一些可以替代它们的软件。

- Gimp
  （Linux 或 OS X：www.gimp.org Windows：http://gimp-win.sourceforge.net/stable.html），免费
- Acorn
  （OS X：http://flyingmeat.com/acorn/）
- Pixelmator
  （OS X：www.pixelmator.com）
- Paint.NET
  （Windows：www.getpaint.net），免费
- PaintShop Pro
  （Windows：www.corel.com）

每款程序都有其自身的使用方式，因此你的选择取决于你的使用偏好。

5.4　保存图像

创建图像后需要保存图像，这一过程是在图像的视觉质量与文件大小之间寻找平衡的艺术。

如果你的计算机上没有安装 Photoshop，可以先使用试用版。

1. Adobe Photoshop

Photoshop 在 File 菜单中提供了 Save for Web（存储为 Web）命令。它让用户可以从视觉上对比原始图像和一至三个优化后的版本，还可以看到对文件大小和下载时间的影响。

2. 使用 Photoshop 的 Save for Web 命令的步骤

(1) 打开 Photoshop，创建图像。或者打开现有的图像，并通过剪裁、调整大小和编辑，为发布做好准备。

(2) 选择 File → Save for Web（在之前的版本中称为 Save for Web & Devices）。出现 Save For Web 对话框，参见图 5.4.1。

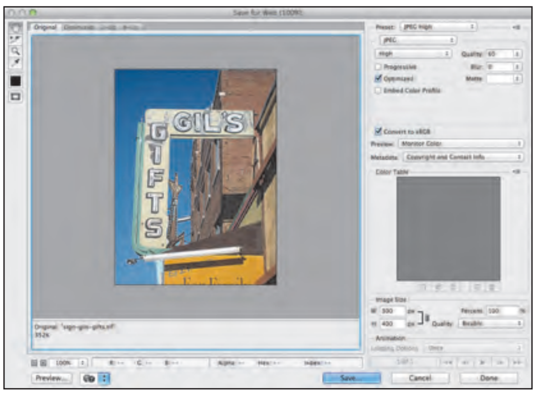


图 5.4.1　Save for Web 对话框默认对应 Original 选项卡，显示原始图像

(3) 点击 2-Up（2 联）选项卡查看一个优化后的版本，或点击 4-Up（4 联）选项卡查看三个优化后的版本，参见图 5.4.2。

(4) 根据需要，点击一个优化后的版本，参见图 5.4.2。



图 5.4.2　选择 4-Up 选项卡（上），可以比较原始图像与你所选择的三个优化方案。这里选择了左下方的图，从而可以修改其设置（图 5.4.3）

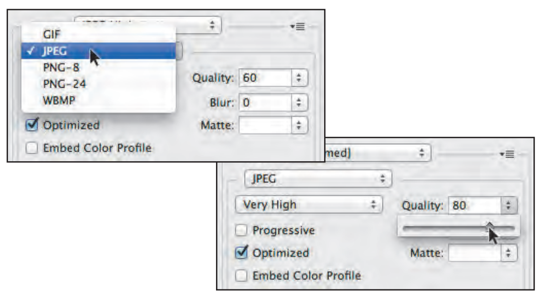


图 5.4.3　Save for Web 对话框的右侧是改变优化设置的控件。可以从下拉菜单（上）中选择图像格式，然后通过直接输入一个数值或者拖动滑动条（下）指定图像的质量。数值越大，质量就越高，而文件也越大。此外，还可以在 Quality 左侧的下拉菜单中选择预设值，如“Very High”（非常高）。尽管我们通常认为 JPEG 是保存照片的最佳格式，但如果质量设的太低，大块的颜色（如本图中的蓝天）就会变得有些模糊不清

(5) 选择希望采用的格式，参见图 5.4.3。

通常，在计算机上创建的图像（包括标志、图表、图形、线条艺术，以及任何包含由单色和锐利细节构成的大块区域的图像）应保存为 PNG-8 格式，参见图 5.4.1。

具有连续色调的图像（如照片）应保存为 JPEG 或 PNG-24 格式。

(6) 调整其他设置选项，直到在图像质量可接受的条件下获得最小的文件，参见图 5.4.4。

(7) 点击 Save。选择一个目录，并对新文件进行命名。它将自动包含所选格式的扩展名（而且，这样通常就不会覆盖原始图像）。

输出 PNG 图像的步骤相同（参见图 5.4.5 和图 5.4.6）。

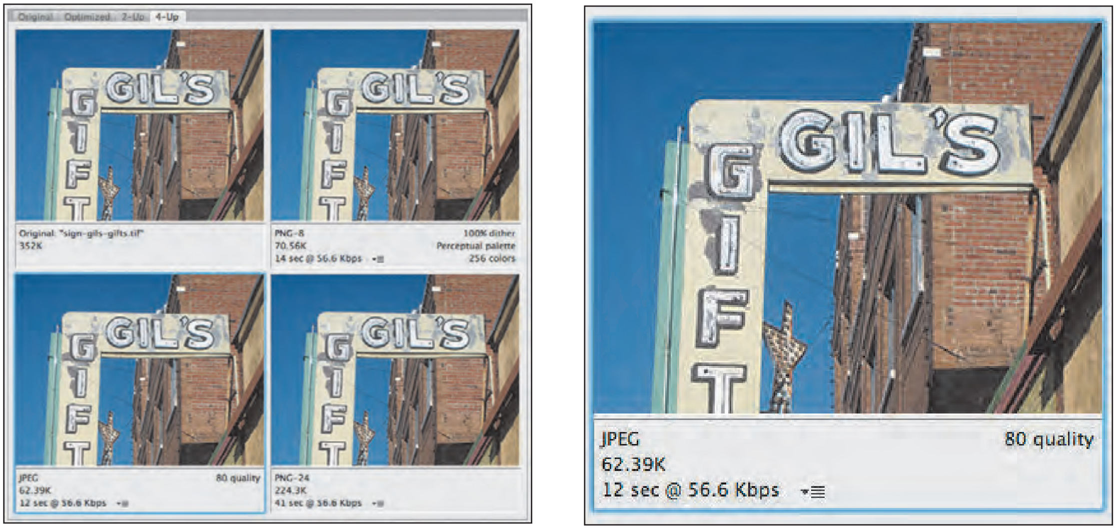


图 5.4.4　这里的修改反映的是左下方的区块。如果保存它，其大小就是 62.39K，尽管在图 5.4.2 中它只有 19.44K，但现在它的边缘更锐利了，这种质量上的提升是值得的。这里还修改了其他两个区块的设置。PNG-8（右上）的压缩让图像的像素化程度加深了，而且文件大小（70.56K）也比 JPEG 要大。PNG-24（右下）保持了很高的质量，但文件大小（224.3K）也大得多。显然，JPEG 是保存照片的最佳选择。另见彩插

提 示　应该使用 RGB[1]模式创建图像，而不是 CMYK[2]模式（用于印刷）。

提 示　在优化和输出图像时，没有所谓的唯一正确或者错误的设置，记住你的主要目标是在保留可以接受的图像品质的前提下让图像尺寸尽可能小。

① R、G、B 分别表示 Red（红）、Green（绿）、Blue（蓝）。RGB 模式又称三原色光模式。——译者注

② C、M、Y、K 分别表示 Cyan（青）、Magenta（品红）、Yellow（黄）、Black（黑）。CMYK 模式又称印刷四分色模式。——译者注

提 示 压缩图像文件大小的工具有很多（甚至可以对已经在图像编辑器中优化过的图像作进一步压缩）。其中有的运行在你的机器上，如 ImageOptim（http://imageoptim.com，只有 OS X 版）和 JPEGmini（www.jpegmini.com），有的则是基于 Web 的，如 www.smushit.com。可以在网上搜索“image optimization tools”（图像优化工具），找到更多这样的工具。推荐使用其中的某个工具。

提 示 如果不确定应该选择哪种格式，可以比较两种优化结果，看看哪种格式的压缩效果更好。另外，不同的编辑器生成的优化图像的尺寸可能不同。



图 5.4.5 这个图像（已被放大）有很多平铺的颜色及文字，它们需要保持锐利。可以看到，PNG-8 格式（左下）对图像的压缩是最好的，文件大小只有 5.5K。如果减少颜色的数量，其大小还将进一步减小。PNG-24 可以提供更多的颜色，使用这种格式的话，文件大小为 11.17K。使用最高质量的 JPEG 则为 19.76K。如果将 JPEG 的质量降为 50（这里没有显示），文件大小可比 PNG-8 更小一点，但其效果看上去却很可怕。另见彩插

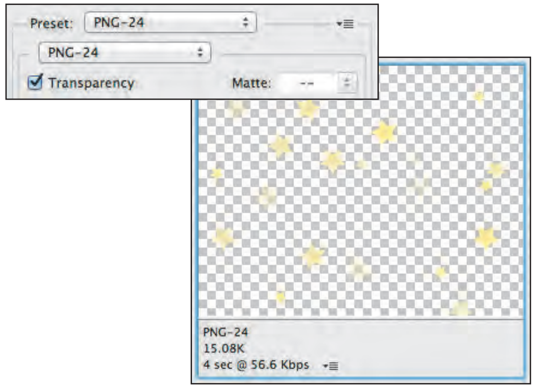


图 5.4.6 对于具有 alpha 透明的图像，如这里的星星图（下），选择 PNG-24 并选中 Transparency（上）。其他的图像编辑器可能将这个选项称为 PNG-32（注意，要在打开 Save for Web 对话框之前设置好图像的透明区域）

提 示 Save for Web 命令会创建新的图像，并保持原始图像不变（除非你在旧图像文件的同一个文件夹使用相同的文件名和扩展名保存新的图像）。

提 示 在 PSD（Photoshop 文档）中，可以让图像的不同部分位于不同的图层，从而可以对它们进行显示和隐藏。通常，在优化后的版本中，只保存了图像的可视图层。Fireworks 可以让 PNG 拥有额外的信息，从而可以保存图层。注意，Photoshop 无法显示这些图层。

5.5 在页面中插入图像

可以在网页中放置各种各样的图像，从标志到照片都可以。当访问者浏览网页时，浏览器会自动加载像图 5.5.1 这样在 HTML 文档中描述的图像，结果参见图 5.5.2。不过，图像加载时间跟访问者的网络连接强度、图像尺寸，以及页面中包含的图像个数相关。

```
...
<body>
<h1>Barcelona's Market</h1>

<img src="corner-market.jpg" />

<p>This first picture shows one of the fruit
→ stands in the <span lang="es">Mercat de la
→ Boquería</span>, the central market that
→ is just off the Rambles. It's an incredible
→ place, full of every kind of food you
→ might happen to need. It took me a long
→ time to get up the nerve to actually take
→ a picture there. You might say I'm kind
→ of a chicken, but since I lived there,
→ it was just sort of strange. Do you take
→ pictures of your supermarket?</p>
</body>
</html>
```

图 5.5.1　这个图像的 URL 只包含文件名，没有路径，因此代表该图像位于与此网页相同的文件夹。关于如何引用不同文件夹中的图像参见第一条提示

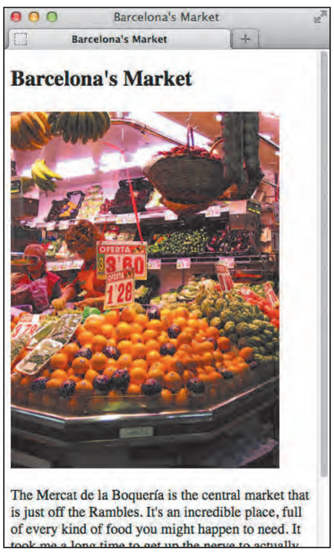


图 5.5.2　图像贴向页面的左侧，与文本的对齐方式一致。使用 float（参见 11.11 节）等 CSS 属性可以改变对齐方式或让文字环绕图像

在页面中插入图像的步骤

(1) 在 HTML 代码中，将光标放在希望图像出现的位置。

(2) 输入 <img src="*image.url*"，其中 *image.url* 指示图像文件在服务器上的位置。

(3) 输入一个空格和 />（或者 >，在 HTML5 中两者均有效）。

提 示　图 5.5.1 中的例子显示了最简单的图像路径形式，即只有文件名。不过，在实践中，为了保持良好的文件组织结构，通常将图像保存在单独的文件夹中。img 标签的 src 属性中的 URL 也应该反映这一路径。假设图 5.5.1 中的页面所在的文件夹还包含一个名为 images 的文件夹，且图像位于 images 文件夹中，则显示该图的 HTML 应为 。更多关于如何引用文件的信息，参见 1.7 节。

提 示　图像必须先上传到服务器上，访问者才有可能看到它们。上传图像跟上传 HTML、CSS、JavaScript 以及其他文件一样，参见 21.3 节。

提 示　不要期望你的访问者会长时间等待页面加载和显示。可以对页面进行测试（别忘了你的连接速度可能比访问者的快）。如果你都等不下去，那么访问者也一样等不下去。另一种办法是为大图创建缩略图，让访问者可以通过链接选择查看大图，查看 6.3 节学习如何创建这种链接。Charles（www.charlesproxy.com）和 Fiddler（http://fiddler2.com）是模拟慢连接的两个不错的工具。

提 示　可以使用 CSS 的 border 简写属性（及相关属性）为图像应用边框样式，这将在第 11 章讲到。旧的浏览器会对带链接的图像自动加上一个边框，可以在 CSS 中指定 img { border: none; } 移除该边框。

5.6　提供替代文本

使用 alt 属性，可以为图像添加一段描述性文本，当图像出于某种原因不显示的时候，就将这段文字显示出来。屏幕阅读器可以朗读这些文本，帮助视障访问者理解图像的内容。HTML5 规范推荐将 alt 文本理解为图像的替代性描述：“一般来说，替代文本是考虑图像未能正常加载的情况下需要呈现的文字。”通常，这意味着 alt 文本可以插入到图像两侧的文本流中，在大多数情况下，它不应是对图像的描述。

提供图像无法显示时的替代文本的步骤

(1) 在 img 标签内，在 src 属性及其值的后面，输入 alt="。

(2) 输入图像出于某种原因没有显示时应该出现的文本（参见图 5.6.1 和图 5.6.2）。

(3) 输入 "。

```
...
<body>
<h1>Barcelona's Market</h1>

<img src="market.jpg" alt="Oranges,
→ bananas, apples, and other fruit abound
→ at the Mercat de la Boquería." />

<p>The <span lang="es">Mercat de la Boquería
→ </span> is the central market that is just
→ off the Rambles. It's an incredible place,
→ full of every kind of food you might
→ happen to need...</p>
</body>
</html>
```

图 5.6.1　我引用了一个不在我的网站的图像（market.jpg），以显示 alt 文本的效果



图 5.6.2　在 Internet Explorer 10 中，替代文本出现在一个带叉的小方块旁边，且两者由一个方框包围。在 Firefox 和 Opera 等其他浏览器中，替代文本是单独出现的。Chrome 和 Safari 不会显示 alt 文本，而是显示缺失图像的图标

提 示　HTML5 规范包含了对在不同场景中如何有效使用 alt 的大量讨论（www.w3.org/TR/html5/embedded-content-0.html#alt），建议读者去看一看。

提 示　如果图像对内容的价值较小，对视障用户来说不太重要，则可以提供空的替代文本，即 alt=""。如果图像与邻近的文本表达的信息相似，也可以将 alt 属性留空。（注意，图 5.6.1 中的例子便符合这一标准，尤其是段落中提到了水果类型的情况。）

提 示　不要用 alt 文本代替图像的 caption。在这种情况下，应考虑将 img 放入一个 figure 元素，并添加一个 figcaption 元素。4.4 节有一个这样的例子。

提 示　如果图像是页面设计的一部分，而不是内容的一部分，则应使用 CSS background-image 属性引入该图像，而不是使用 img 标记。参见 10.10 节。

为什么图像无法显示

图像无法显示的原因有很多种。也许是因为你在 src 属性中填写的 URL（参见图 5.6.1）是错误的，也许是因为你忘了将图像上传到服务器。其他原因则超出了你的控制范围，如访问者的网络连接状况可能很糟糕。此外，你知道可以让浏览器不加载图像吗？大多数浏览器都可以在偏好设置中设定是否加载图像。通常，为了加快页面加载速度，用户可能选择这一选项。使用手机或平板电脑的用户，为了防止侵吞太多流量，也有可能选择这一选项。

5.7　指定图像尺寸

有时，加载网页会先看到文本，等一小段时间以后图像开始加载时，文本跳到图像周围，留出可容纳图像的空间。出现这种现象，是因为没有在 HTML 中指定图像的尺寸。如果用户使用旧浏览器或者网络连接速度慢，就容易出现这种情况。

如果指定图像的尺寸，浏览器就可以预留空间，在图像加载的同时让文本显示在周围，保持布局的稳定。

可以通过浏览器或图像编辑软件获取图像的精确尺寸。

1. 在浏览器中查看图像尺寸

(1) 右击图像，出现背景弹出菜单，参见图 5.7.1。

(2) 选择 View Image Info（查看图像信息，具体选项取决于所使用的浏览器），出现的框中会以像素为单位显示图像的尺寸，参见图 5.7.2。

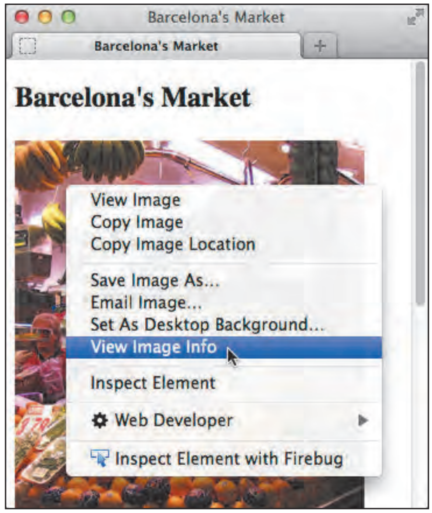


图 5.7.1　在浏览器中右击图像，出现背景弹出菜单。浏览器会提供检查图像、显示属性或获取尺寸的方式

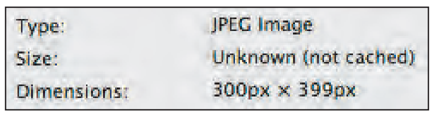


图 5.7.2　这个框（它的外观取决于所用的浏览器）以像素为单位显示了图像的尺寸

2. 在 Photoshop 中查看图像尺寸

(1) 在 Photoshop 中打开图像。

(2) 选择 Image → Image Size（如图 5.7.3 所示）。出现 Image Size 对话框（参见图 5.7.4）。

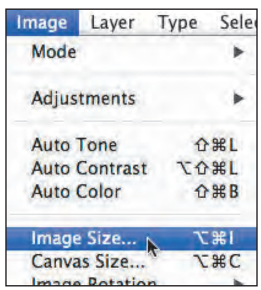


图 5.7.3　在 Photoshop 中，从 Image 下拉菜单中选择 Image Size

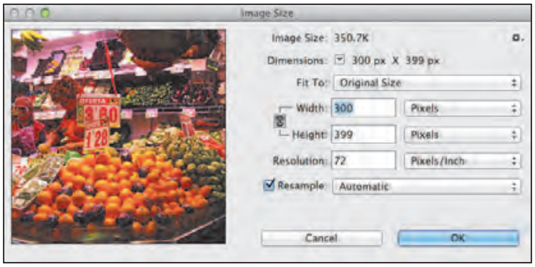


图 5.7.4 Image Size 对话框显示图像为 300 像素 × 399 像素（可以改变这个对话框的大小，确保在 Width 或 Height 旁边的下拉菜单中选择的单位为 Pixels，再输入新的 Width 值，Height 值会根据图像的比例自动计算出来。更多信息参见最后一条提示）

3. 在 HTML 中指定图像尺寸

(1) 使用“在浏览器中查看图像尺寸”或“在 Photoshop 中查看图像尺寸”中介绍的方法，确定图像的尺寸。

(2) 在 img 标签中，src 属性的后面，输入 width="*x*" height="*y*"，采用第 (1) 步中确定的值，以像素为单位指定 *x* 和 *y*（分别代表图像的宽度和高度）的值，参见图 5.7.5。

```
...
<body>
<h1>Barcelona's Market</h1>

<img src="corner-market.jpg" width="300"
→ height="399" alt="Oranges, bananas, apples,
→ and other fruit abound at the Mercat de la
→ Boquería." />

<p>The <span lang="es">Mercat de la Boquería
→ </span> is the central market that is just
→ off the Rambles. It's an incredible place
→ ...</p>

</body>
</html>
```

图 5.7.5 通常我们会在 HTML 中明确指定图像的高度和宽度，这样浏览器就不必花时间来判断图像的尺寸，从而更快地将图像显示出来。但有一种情况我们应该忽略 width 和 height 属性，即显示响应式图像的情况，我们会在 12.2 节讨论这个问题

提 示 width 和 height 属性不一定要反映图像的实际尺寸。

提 示 如果有几个尺寸相同的图像，可以通过样式表同时设置它们的高度和宽度。当然，单个图像也能使用这种方法。

提 示 在浏览器中，如果在单独的窗口中打开图像，就可以看到图像的尺寸，参见图 5.7.6。

提 示 在 Photoshop 中可以选中整个图像，再在 Info（信息）面板中查看图像的尺寸。

提 示 如果使用如图 5.7.4 所示的 Image Size 对话框修改图像的尺寸，要确保 Resample（重新取样）选项为选中状态。Resolution（分辨率）选项与 Web 图像无关。5.9 节介绍了另外一种在 Photoshop 中改变图像尺寸的方法。



图 5.7.6 如果在单独的窗口中打开图像，或者将图像拖入单独的窗口，标题栏里就会显示图像的尺寸

5.8　在浏览器中改变图像的尺寸

通过为图像指定新的高度和宽度（以像素为单位），可以改变图像显示的尺寸，参见图 5.8.1、图 5.8.2 和图 5.8.3。这样做可以在保持所有显示屏上图像尺寸相同的情况下，让使用 Retina 显示屏的用户看到锐利的图像。具体参见补充材料“创建和缩放为 Retina 显示屏准备的图像”。

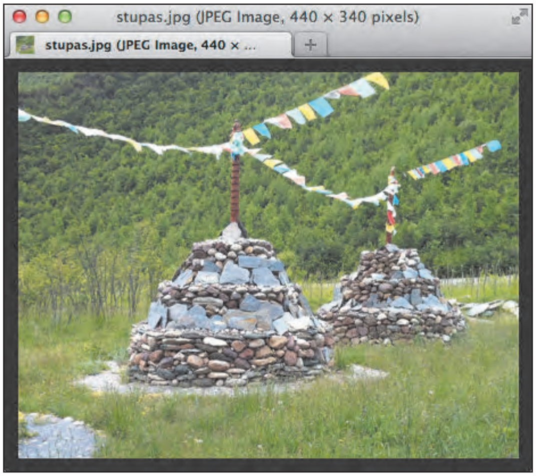


图 5.8.1　这张图在创建时使用的是双倍尺寸（440 像素 ×340 像素），但在页面中显示的尺寸只有实际尺寸的一半（即 220 像素 ×170 像素），这样就能提升 Retina 显示屏及其他高像素密度显示屏上图像的锐利性（黑色的轮廓线来自于浏览器，而不是图像本身）

在浏览器中改变图像尺寸的步骤

(1) 输入 `<img src="image.url"`，其中 *image.url* 是图像在服务器上的位置。

(2) 输入 `width="x" height="y"`，其中 *x* 和 *y* 分别是希望设定的宽度和高度（以像素为单位）。

(3) 根据需要，添加任何其他的图像属性，最后输入 `/>`。

```
...
<figure>
    <img src="stupas.jpg" alt="Two Stupas"
    ➝ width="220" height="170" alt="" />

    <figcaption>These stupas in Yunnan,
    ➝ China, are Buddhist monuments used as
    ➝ a place for worship.</figcaption>
</figure>
...
```

图 5.8.2　调整 height 和 width 属性均为原来的一半，由于图像的高度和宽度比例保持不变，图像就不会失真



图 5.8.3　显示的图像只有原图的一半大小。不过要注意，加载它需要花的时间跟以前是一样的。毕竟，用的是同一个文件

提示　除了为 Retina 显示屏准备的图像，使用 width 和 height 属性改变图像显示在网页中的大小是一种快捷但有些丑陋的做法。由于文件本身并未改变，因此访问者容易有受骗的感觉——通常用这种方法缩小的图像总是比实际这个尺寸的图像加载得更慢。较好的做法是使用图像编辑器改变图像的尺寸。

提示 Thomas Fuchs（http://retinafy.me） 和 Daan Jobsis（http://blog.netvlies.nl/designinteractie/retina-revolution/）讨论过一种有趣的技术，通过这项技术可以创建为 Retina 显示屏准备的图像，同时保持文件大小控制在一个合理的范围之内。其做法为：让图像尺寸变为两倍，但使用高压缩设置（在 Photoshop 中，就是选择低的图像质量）。当以一半大小显示图像时，不容易察觉到高压缩产生的失真效果的影响。实际的效果取决于图像的内容。

创建和缩放为 Retina 显示屏准备的图像

你可能在一些人提到苹果的 iPhone、iPad 和 MacBook 时听说过 **Retina 显示屏**。Retina 显示屏是什么呢?

设想你仅使用点来作画，就像克洛德·莫奈或乔治·修拉[①]的画。然后再设想在同样大小的画布上描绘相同的画，只是每一个点都用四个点来代替。第二张画能呈现更多的细节，同时也更难看清单独的点。

苹果的 Ritina 显示屏就像第二张画。在相同的空间里，它拥有的像素数量是普通显示屏的像素数量的四倍（如图 5.8.4 所示），因此图像会更锐利。专业的说法是，它的**每英寸点数**（PPI）更多，或者说像素密度更大。除了苹果，还有其他一些公司的设备也使用了高像素密度显示屏。

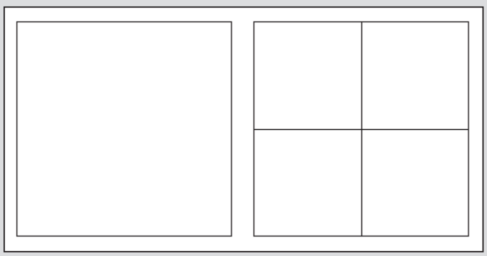


图 5.8.4 这张图有助于从视觉上理解 Retina 显示屏（右）是如何使用四个像素取代大多数显示屏（左）的一个像素的。这并非缩放，实际的像素要小得多，我想你应该能理解

需要说明的是，有时需要考虑 Retina 显示屏或类似的其他显示屏，否则图像在浏览器里看起来就会显得模糊。如果你不关心其中的原因，至少应记住，让图像的尺寸扩大为原先的两倍，但仅以一半的尺寸显示它们。

例如，如果你想让图像在所有的显示屏（并非只是 Retina 显示屏）上都是 40×30 的尺寸（此处及以下尺寸单位均为像素），就应该创建 80×60 大小的图像，而代码写作 `<img src="photo.jpg" width="40" height="30" alt="" />`。浏览器会将 80×60 的图像缩小，以 40×30 的尺寸显示（图 5.8.1 至图 5.8.3 展示了这种方法）。

① 克洛德 · 莫奈（Claude Monet）和乔治 · 修拉（George Seurat）都是印象派画家，他们的作品往往充满了细腻缤纷的小点。——译者注

其中的原理是这样的，在 80×60 的图像中，像素总数为 4800，是原先的四倍（40×30 为 1200 像素）。这样，Retina 显示屏就可以显示这些额外的像素，让图像看起来更锐利。如果使用 40×30 的图像，Retina 显示屏就会拉伸这些像素以填充其对应的空间，导致锐利程度降低。每个图像的情况都不一样。

代码编写人员和设计师通常不会让每个图像的分辨率都扩大为原先的两倍，具体做法取决于开发人员自己。如果图像特别重要（如摄影师的作品集网站），强烈推荐这种做法。这样图像的大小可能会变大（要了解如何避免这种情况，参见第二条提示），而且双倍分辨率的图像会消耗设备更多的内存（这主要是移动设备的问题）。建议至少在一个移动设备上做好测试。

◎ 图标字体与 SVG

图标字体与 SVG 在缩放时都不会导致失真。对于单色的图标，建议尽可能地使用图标字体，而非使用图像（参见 13.2 节）。对标识和其他非照片类图像，可以考虑使用 SVG（关于 SVG 的说明，参见 5.1 节的补充材料）。

5.9　在图像编辑器中改变图像的尺寸

大多数图像对网页来说都太大了。一幅用于打印的图像可能有 1800 像素宽（以 300 dpi 打印，有 6 英寸宽，约 15 厘米），而用于网页的图像很少需要超过 600 像素，通常还要小得多。

使用图像编辑器既可放大图像，也可缩小图像。但是图像放大后质量通常会变差，看起来非常明显。此外，放大图像还会增加图像文件的大小，导致页面加载时间变长。

用 Photoshop 改变图像尺寸

(1) 在 Save for Web 窗口右下角的 Image Size 部分，点击 W（宽度）框或 H（高度）框，如图 5.9.1 所示。

(2) 以像素为单位输入宽度或高度，或输入百分数，再按一下 Tab 键，改变图像的尺寸，如图 5.9.2 所示。

(3) 可以继续调大或调小，直到满意为止。在按下 Save 之前，图像不会重新取样。

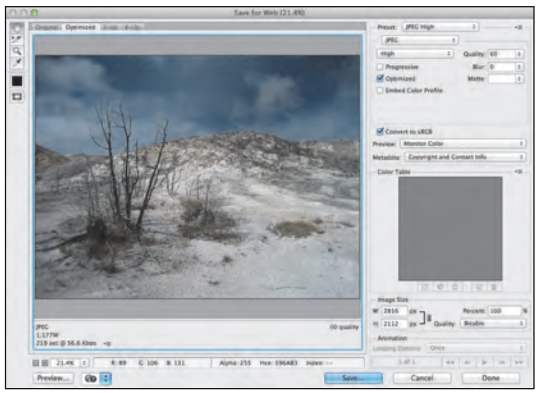


图 5.9.1　原始图像是用笔者的数码相机在默认设置下拍摄的，其尺寸为 2816 像素 ×2112 像素，对于常见的网页来说太大了，将图像压缩为高质量 JPEG 后仍然有 1.177MB。要是你在自己的网站上使用这样一张图像，肯定会让你的访问者惟恐避之不及

提 示　在进入 Save for Web 之前，也可以通过 Image（图像）菜单下的 Image Size（图像大小）命令改变图像的尺寸，参见图 5.7.3 和图 5.7.4。

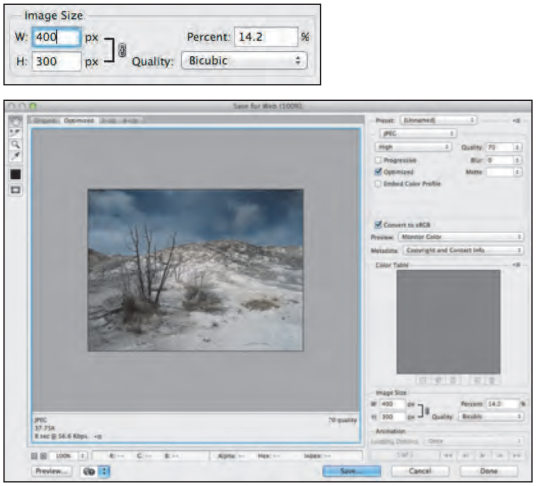


图 5.9.2　在 W 字段中输入新的宽度（这里是 400 像素），H 字段中的高度值会按照原始比例做相应改变。缩小后的图像（下图）在页面中很合适，通过改变压缩设置，文件大小降到了 37.75K

提 示　减小图像尺寸的另一个好办法是将图像上不需要的区域裁剪掉。

5.10　为网站添加图标

我们在浏览器选项卡（参见图 5.10.1）、历史记录、书签页、收藏夹和地址栏中看到的（与网站相关）的小图标称为 favicon，这个词是 favorites icon（收藏夹图标）的简称。我们创建的图标至少应该为 16×16（所有图标大小单位均为像素），不过，无论如何，浏览器都会尝试尽力加载。

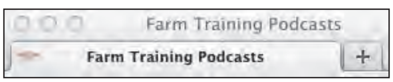


图 5.10.1　相较于浏览器的其他位置，favicon 更多地显示在标签页上（如本图所示）。Internet Explorer 是唯一将它显示在地址栏上 URL 之前的浏览器。由于浏览器通常将该图标显示在灰色或其他颜色的背景上，因此有必要将图标的背景设为透明的（如图 5.10.2 所示）

此外，还可以创建一个或多个触屏图标，即在苹果设备及其他触屏设备上将网站添加到主屏幕上时要显示的图标。苹果规定，iPhone 和 iPod touch 的图标大小为 57×57 或 114×114（对于 Retina 显示屏），iPad 的图标大小为 72×72 或 144×144（对于 Retina 显示屏）。Android 操作系统也支持这些图标。

为网站添加图标

(1) 创建一个 16×16 的图像，并保存为 ICO 格式，文件名为 favicon.ico，如图 5.10.2 所示。作为可选步骤，为 Retina 显示屏创建一个 32×32 的图像。ICO 文件允许在同一个文件中包含多个不同尺寸的同名文件。

(2)（推荐）为触屏设备至少创建一个图像（如图 5.10.3 所示），并保存为 PNG 格式。如果只创建了一个，将其命名为 apple-touch-icon.png。如有需要，还可以创建其他的触屏图标。

(3) 将图标图像放在网站的根目录里。浏览器会自动在根目录寻找这些特定的文件名，找到后就将图标显示出来。

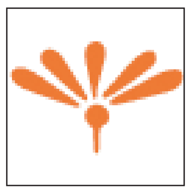


图 5.10.2　在现实环境中，favicon 比这里显示的要小，它们只有 16 像素 ×16 像素



图 5.10.3　将网站从 Safari 添加到 iOS 设备的主屏幕上会用到 `apple-touch-icon`

提示　有很多 ICO 格式图像编辑器，如 X-Icon Editor（http://xiconeditor.com）。此外，还有一些 Photoshop 插件（可以在网上搜索）。

提示　也可以将图标文件放在根目录以外的其他文件夹，不过这样的话，就需要在 HTML 中添加 `link` 元素，从而让浏览器得以找到这些图标（参见补充材料）。

提示　HTML5 Boilerplate（h5bp.com）里提供了各种尺寸的触屏图标的示例。

关于 favicon 的更多信息

关于创建 favicon 的更多信息，参见 www.netmagazine.com/features/create-perfect-favicon。（注意，其中的一些信息已经过时了）。此外，Thomas Fuchs 有一篇关于 favicon 的文章，见 http://davidwalsh.name/retina-favicons。

第 6 章

链接

6

本章内容

- 创建指向另一个网页的链接
- 创建锚并链接到特定的锚
- 创建其他类型的链接

链接是万维网的命脉。没有它们，每个页面都只能独立存在，同其他所有页面完全地分开。

链接有两个主要的部分：目标和标签。使用**目标**（destination）可以指定访问者点击链接时会发生什么。可以创建链接进入另一个页面，在页面内跳转，显示图像，下载文件，呼叫电话，等等。不过，最常见的是连接到其他网页的链接，其次是连接到其他网页特定位置（称为**锚**，anchor）的链接。目标是通过编写 URL 定义的，通常只能在（桌面）浏览器的状态栏中看到。

链接的第二个部分是**标签**（label），即访问者在浏览器中看到或在屏幕阅读器中听到的部分。激活标签就可以到达链接的目标。例如，航空公司网站上可能有这样的链接标签：预订航班。标签可以是文本、图像或二者兼有。浏览器通常会将标签文本默认显示为带下划线的蓝色文字。通过 CSS 可以很容易地改变这一样式。

6.1 创建指向另一个网页的链接

a 元素是创建链接的关键，如图 6.1.1 所示。如果你有多个网页，那么很可能希望创建从一个页面到另一个页面（以及返回）的链接（参见图 6.1.2 ~ 图 6.1.4）。还可以链接到其他网站的页面，可能是你自己制作的，也可能是他人创建的（参见图 6.1.5 ~ 图 6.1.8）。

1. 创建指向另一个网页的链接

(1) 输入 `<a href="`*`page.html`*`">`，其中 *`page.html`* 是目标网页的 URL，如图 6.1.1 所示。

(2) 输入标签文本，也就是默认突出显示的文本，如图 6.1.3 所示，访问者激活它时，就会转到第 (1) 步中所指向的页面。也可以添加一个 `img` 元素替代文本（或同文本一起）作为标签。（参见 6.3 节及补充材料“将缩略图链接到图像”。）

(3) 输入 `</a>` 结束链接定义。

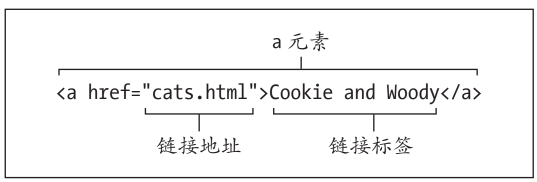


图 6.1.1 每个链接都有一个目标 URL 地址（由 `href` 属性指定）和一个标签。这个例子中的标签是文本，不过标签也可能是图像或者文本和图像的组合

```
...
<body>

<h1>Cookie and Woody</h1>
<img src="img/cookie.jpg" width="143" height="131" alt="Cookie" />

<img src="img/woody.jpg" width="202" height="131" alt="Woody" />

<p>Generally considered the sweetest and yet most independent cats in the <a href="pioneer-valley.
→ html">Pioneer Valley</a>, Cookie and Woody are consistently underestimated by their humble
→ humans.</p>

</body>
</html>
```

图 6.1.2　由于在 href 属性中只有文件名（没有路径），文件 pioneer-valley.html 必须与包含这个链接的网页位于同一个目录。否则，当用户激活该链接时，浏览器将找不到 pioneer-valley.html



图 6.1.3　当访问者指向链接，在大多数浏览器中，目标 URL 会显示在状态栏中。（图 6.1.2 中的示例假设两个网页都位于 www.fictitious-site.com）如果用户激活该链接……

可以创建指向另一个网站的页面的链接，例如：`<a href="http://www.site.com/directory/page.html" rel="external">Label text</a>`（参见图 6.1.5 ~ 图 6.1.8）。将 href 的值替换为目标URL地址，rel 属性是可选的，即便没有它，链接也能照常工作。但对于指向另一网站的链接，推荐包含这个值。它描述包含链接的页面和链接指向的页面之间的关系。它也是另一种提升 HTML 语义化程度的方式。搜索引擎也会利用这些信息。此外，还可以对带有 rel="external" 的链接添加不同的样式，从而告知访问者这是一个指向外部网站的链接。（参见 9.8 节。）

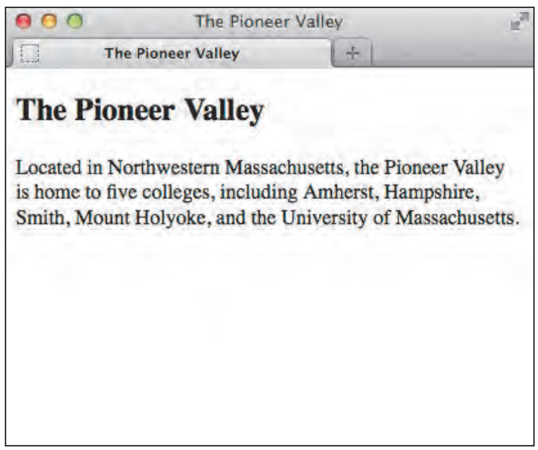


图 6.1.4　……与这个目标 URL 相关联的页面就会显示在用户的浏览器中

通过键盘进行导航

可以通过键盘对网页进行导航。实际上，有的人由于有一些肢体障碍，无法使用鼠标这样的设备，因此不得不通过键盘对页面进行导航。

每按一次 Tab 键，焦点就会转移到 HTML 代码中出现的下一个链接、表单控件或图像映射（每按一次 Shift+Tab，焦点就会向前转移）。这个顺序不一定与屏幕上出现的顺序一致，因为页面的 CSS 布局可能不同。通过使用 HTML 的 `tabindex` 属性，可以改变使用 Tab 键的顺序，但不建议使用，因为在大多数情况下没必要这么做，而且这种方法已经过时了。这种方法可能导致屏幕阅读器用户迷失方向。（14.9 节演示了 `tabindex` 的一个有帮助的用法。）

对内容添加标记的时候，应注意保证转移焦点的顺序是符合逻辑的。应对自己的页面使用 Tab 键进行测试，就像用户的操作一样，再对 HTML 作相应的调整。（注意，如果你使用的是 Mac，这一功能可能被禁用了。如果要开启这一功能，可以在网上搜索“Enabling keyboard navigation in OS X browsers”（在 OS X 浏览器中激活键盘导航）。

```
...
<body>

<h1>The Glory of Cats</h1>

<p><a href="http://en.wikipedia.org/wiki/
→ Cat" rel="external" title="Cat entry
→ on Wikipedia">Cats</a> are wonderful
→ companions. Whether it's a bottle cap,
→ long string, or your legs, they always
→ find something to chase around.</p>

<p>In fact, cats are so great they even have
→ <a href="http://www.catsthemusical.com"
→ rel="external" title="Official site of
→ Andrew Lloyd Webber's musical">their own
→ musical</a>. It was inspired by T.S. Eliot's
→ <cite>Old Possum's Book of Practical Cats
→ </cite>.</p>

</body>
</html>
```

图 6.1.5 如果要创建的链接是指向其他网站的，应使用绝对 URL。绝对 URL 由 http://、主机及完整路径（如果需要的话）组成。同 `rel` 一样，`title` 属性也是可选的（关于 `cite` 元素的用法，参见第 4 章）

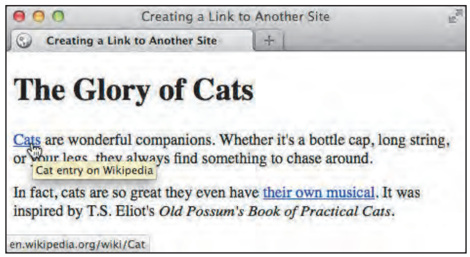


图 6.1.6 同指向站内页面的链接一样，访问者将鼠标移到指向其他网站的链接上时，目标 URL 会出现在状态栏里，`title` 文字（如果指定了的话）也会显示在链接旁边。当访问者点击链接时……



图 6.1.7 ……目标 URL 指示的页面会显示在访问者的浏览器中

2. HTML 块级链接

目前，你已看过一些将某一元素内的一小部分文本作为链接的例子。实际上，HTML5 几乎允许在链接内包含任何类型的元素或元素组，参见图 6.1.8。例如段落、列表、整篇文章和区块——几乎任何元素都行（但其他链接、音频、视频、表单元素、`iframe` 等交互式内容除外），这些元素大部分为块级元素。使用 HTML 验证器对页面进行测试（参见 20.5 节）可以防止链接中出现不允许包含的元素。

```
…
<a href="giraffe-escapes.html">
    <p>A giraffe escaped from the zoo today,
    → and animals rejoiced worldwide.</p>
    <p>Read more</p>
</a>

… [更多标题] …
```

图 6.1.8　恐怕大家都访问过这样的网站，以文章的一小段内容为链接，指向完整的文章。如果想让这一小段内容和提示（如“阅读全文”）都形成指向完整文章页面的链接，就应使用块级链接。可以通过 CSS 让部分文字显示下划线，或者所有的文字都不会显示下划线

这些块级链接（block-level link）是 HTML5 同 HTML 早期版本有巨大差异的地方。在以前的 HTML 中，链接中只能包含图像、文本短语，以及标记文本短语的元素（如 `em`、`strong`、`cite` 等）。

有趣的是，尽管在以前的 HTML 规范中块级链接是不允许的，但浏览器都支持。这意味着你现在就可以使用它们，而且它们在旧的浏览器和现代浏览器中均能正常工作。不过，使用它们的时候也要小心（参见图 6.1.8 和图 6.1.9）。

有一些可访问性方面的注意事项，特别是涉及不同的屏幕阅读器如何处理块级链接的问题。无障碍访问专家 Derek Featherstone 和 Steve Faulkner 的文章深入探讨了这个问题，参见 http://simplyaccessible.com/article/html5-block-links/ 和 www.paciellogroup.com/blog/2011/06/html5-accessibilitychops-block-links/。他们建议将最相关的内容放在链接的开头，而且不要在一个链接中放入过多内容。Featherstone 指出，随着屏幕阅读器和浏览器逐渐开始官方支持块级链接，可访问性问题可能只是暂时的。

```
…
<body>
<a href="pioneer-valley.html">
    <h1>Cookie and Woody</h1>
    <img src="img/cookiefora.jpg" width=
    → "143" height="131" alt="Cookie" />

    <img src="img/woodygran.jpg" width="202"
    → height="131" alt="Woody" />

    <p>Generally considered the sweetest
    → and yet most independent cats in the
    → Pioneer Valley, Cookie and Woody are
    → consistently underestimated by their
    → humble humans.</p>
</a>
…
</body>
</html>
```

图 6.1.9　不要过度使用块级链接。应该避免这里演示的情况——将一大段内容使用一个链接包起来。尽管这样的链接是有效的 HTML5，但屏幕阅读器有可能将所有这些内容多朗读一次，多读的这些内容可能比访问者本希望听到的链接信息要多得多。因此，最好仅将与链接的含义密切相关的内容放在链接里

一般来说，用的最多的还是第一个例子（参见图 6.1.1）那样简单、传统的链接样式，不过也要知道，也用得上制作精巧的块级链接。

提 示　可以改变标签文本的默认样式（参见第 10 章），甚至可以用图像作为标签（参见 6.5 节）。

提示 href 指 hypertext reference（超文本引用）。通常，对指向站内网页的链接使用相对 URL，对指向其他网站页面的链接使用绝对 URL。更多信息参见 1.7 节。

提示 仅指定路径，省略文件名，就可以创建指向对应目录下默认文件（通常为 index.html）的链接：www.site.com/directory/。如果连路径也省略，就指向网站的默认（首）页：www.site.com。

提示 在 URL 中应该全部使用小写字母，除非指向的页面或目录名称中含有大写字母。对于你自己的网站，应使用小写字母命名所有的文件夹和文件，并与链接 URL 对应，相关内容参见 1.6 节。

提示 不要让链接的标签太长。如果链接是句子的一部分，应在链接的标签定义中仅保留关键字。

提示 不管怎样，应避免使用"点击此处"作为标签。这种类型的链接文本在万维网上实在是太常见了，它会破坏网站的可用性和可访问性，从而对网站的拥有者产生不利的影响。当用户快速扫过页面上的链接（无论是通过屏幕还是通过屏幕阅读器）时，会发现"点击此处"缺乏上下文（"点击此处？为什么？"）。它对激活链接几乎不会产生激励，而且依赖于访问者阅读链接周围的文字，并寄希望于这些文字可以解释链接的目的。不难理解，访问者通常更倾向于跳过这样的链接。此外，正如本章开头提到的，"点击"一词并不适用于用户触发链接的所有方式。相反，应该使用文本中已经存在的关键字对链接进行标识。例如，应使用"了解我们的销售情况"，而不是"点击此处了解我们的销售情况"。

提示 如果你想为块级链接添加背景、边框、外边距、内边距等样式，需要在样式表中为链接设置 `display: block`（诚然，这些内容有些超前了）。关于 CSS 中的 `display` 属性，参见 11.6 节。

提示 尽管这里不会讲解图像映射的用法，但你应知道它们是用来为单张图像的一个或多个区域添加链接的。可以在网上搜索"HTML image maps"（HTML 图像映射）了解有关图像映射的更多信息。

提示 维基百科上有一份 `rel` 值的列表（http://microformats.org/wiki/existing-rel-values），该列表还在持续更新中。

提示 一定要在网站的每个页面包含指向网站各主要板块（包括首页）的导航。这可以让访问者自由地浏览网站，不管他们是直接访问网站还是通过其他网站的链接访问。你无法知道访问者会从哪里进入你的网站，可能是通过指向网站内页的"深度"链接，因此你应该让他们可以从那里开始访问网站的其余部分。

6

如何使用 target 属性，以及为何不要在大多数情况下使用该属性

可以将链接设置为在新的窗口或标签页（取决于所用的浏览器）中打开，但我们将其看做一种不好的实践，不推荐使用。这样说是有理由的。

首先，应该让用户决定是否在不同的窗口或标签页打开链接，而不是让 HTML 开发人员决定。否则，我们就支配了用户的浏览行为。

也有一些可用性和可访问性方面的考虑。缺乏经验的用户在激活一个链接却没有在当前窗口看到结果时可能会有一些疑惑。使用浏览器并非对所有人都是一件容易的事。我曾向不同年龄段的人展示标签页，他们之前并不知道可以同时打开多个页面。类似地，屏幕阅读器等辅助设备的用户将不得不花费精力去寻找新的窗口或标签页，前提还是他们明确知道是哪个窗口或标签页加载了新的内容。

如果所有这些并未说服你避免将链接设置为在新窗口和标签页打开，或者你的老板或客户不赞同你不使用它们的理由，可以看看如何创建这种链接：在链接定义中输入 `target="`*`window`*`"`，其中 *window* 是应该显示相应页面的窗口的名称（由你自己选择）。

例如，`<a href="some-page.html" `**`target="doodad"`**`>Some page</a>` 会在名为 doodad 的新窗口或标签页中打开 some-page.html。

如果让多个链接指向同一个窗口（即使用同一个名称），链接将都在同一个窗口打开。或者，如果你希望链接总是在不同的窗口或标签页打开（即使多次激活同一个链接），就使用 HTML 预定义的名称 `_blank`（`target="_blank"`）。

不过我还是不推荐你这样做，能免则免。

`target` 还有一种用法，就是在 `iframe` 中打开链接。可以用同样的方法编写 `target`，只是其值应与 `iframe` 的 `id` 对应。你很少有机会用到这个，特别是在通常不推荐使用 `iframe` 的情况下（不过它们偶尔会有用武之地）。关于 `iframe` 元素的更多信息，参见 https://developer.mozilla.org/en/HTML/Element/iframe。

6.2　创建锚并链接到锚

通常，激活一个链接会将用户带到对应网页的顶端。如果要想用户跳至网页的特定区域，可以创建一个**锚**，并在链接中引用该锚，参见图 6.2.1、图 6.2.2 和图 6.2.3。FAQ 页面可能是最常使用锚链接的页面。

1. 创建锚的步骤

(1) 将光标放在希望用户跳转至的元素的开始标签里。

(2) 输入 `id="`*`anchor-name`*`"`，其中 *anchor-name* 是在内部用来标识网页中这部分内容的文字。一定要在元素名称和 `id` 之间保留一个空格，例如 `<h2 id="features">`。

2. 创建链接到特定锚的链接

(1) 输入 `<a href="#`*`anchor-name`*`">`，其中 *anchor-name* 是目标的 `id` 属性值（参见“创建锚的步骤”中的第 (2) 步，例如 `<a href="#features">`。

(2) 输入标签文本，即突出显示的文本（通常默认显示为带下划线的蓝色文字），也即用户激活它时将用户带到第 (1) 步中引用的区

域的文本。跟通常一样，也可以将图像用做标签。

(3) 输入 结束对链接的定义。

```
...
<body>

<article>
    <header>
      <h1>Frequently Asked Questions (FAQ)</h1>

      <nav>
        <ul>
          <li><a href="#question-01">Can an id have more than word?</a></li>
          <li><a href="#question-02">Can visitors bookmark anchor links?</a></li>
          <li><a href="#question-03">My anchor link isn't working. What am I doing wrong?</a></li>
          ...
        </ul>
      </nav>
    </header>

    <h2 id="question-01">Can an id have more than word?</h2>
    <p>Yes, your ids can have more than one word as long as there are no spaces. Separate each word
    ➝ with a dash instead.</p>

    <h2 id="question-02">Can visitors bookmark anchor links?</h2>
    <p>Yes, they can! And when they visit that link, the browser will jump down to the anchor as
    ➝ expected. Visitors can share the link with others, too, so all the more reason to choose
    ➝ meaningful anchor names.</p>

    <h2 id="question-03">My anchor link isn't working. What am I doing wrong?</h2>
    <p>The problem could be a few things. First, double-check that you added an id (without "#") to
    ➝ the element your link should point to. Also, be sure that the anchor in your link <em>is</em>
    ➝ preceded by "#" and that it matches the anchor id.</p>

    ...
</article>

</body>
</html>
```

图 6.2.1　每个以 # 开头的链接 href 值都指向拥有相应 id（不含 #）的元素。例如，... 指向 <h2 id="**question-03**">...</h2>。可以为任何元素指定 id，只要任何给定的 id 在一个页面中只存在一次（参见 3.14 节）。这个例子还让你提前看到了无序列表（ul）的应用。无序列表是目前万维网上使用频率最高的列表类型（第 15 章将深入讨论列表）

提 示　为每一个锚 id 赋一个有意义的名称可以增强 HTML 文档的语义丰富度。换句话说，避免使用像 anchor3 这样的 id。

提 示　如果锚位于另一个文档，就使用 <a href ="**page.html#anchor-name**"> 引用该区域。（在 URL 和 # 之间没有空格。）如果锚位于另一台服务器上的页面，则需输入 <a href="**http://www.site.com/directory/page.html#anchor-name**">（没有空格）。

提 示　如果锚位于页面的底部，且它下面的内容的高度小于浏览器的可视区域的高度，那么它可能不会显示在窗口的顶部，而是显示在中间。

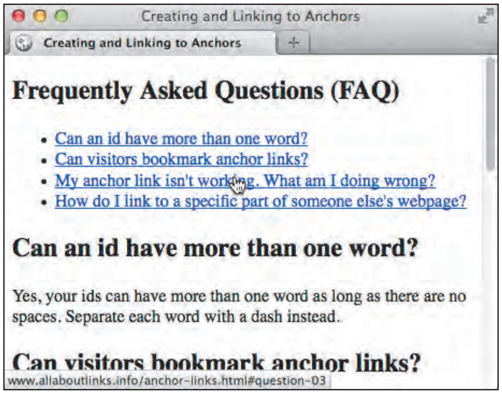


图 6.2.2　在桌面浏览器中，当访问者将鼠标指向引用锚的链接时，URL 和锚名称会显示在状态栏中（在窗口的左下角）

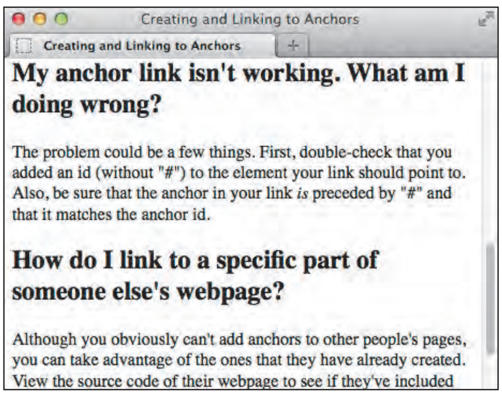


图 6.2.3　当访问者激活该链接时，锚引用的页面特定部分就会显示在浏览器窗口的顶部。这个页面比较搞笑，它链接到了打不开锚链接的情况下应该如何处理的信息

6.3　创建其他类型的链接

并非只能创建指向其他网页的链接，其实可以创建指向任何 URL 的链接——RSS 源、图像、希望访问者可以下载的文件、电子邮件地址、电话号码等（参见图 6.3.1）。

```
...
<h1>Other Types of Links</h1>

<p>There are lots of different kinds of
→ links that you can create on a webpage.
→ Following are some examples.</p>

<h2>Images</h2>
<p>You can link directly to <a href=
→ "img/blueflax.jpg">a photo</a> or
→ even make links out of photos. For
→ example, the following image is linked
→ to a flowers photo gallery page.
→ <a href="gallery-flowers.html" title=
→ "More flower images"><img src="img/
→ blueflax.jpg" width="165" height="105"
→ alt="Blue Flax" /></a></p>

<h2>Other Assets</h2>
<p><a href="media/piano.mp3">Listen to
→ tickling of the ivories</a> (MP3, 1.3 MB)
→ or <a href="media/paddle-steamer.mp4">
→ watch a paddle steamer</a> (MP4, 2.4 MB).
→ These link directly to the files (handy for
→ downloading).</p>

<h2>Email Addresses</h2>
<p>Send feedback to <a href="mailto:
→ someone@somedomain.com">someone@
→ somedomain.com</a>.</p>

<h2>Phone Numbers</h2>
<p>Call now for free things!
→ <a href="tel:+18001234567">1
→ (800) 123-4567</a></p>
...
```

图 6.3.1　可以创建指向各种类型 URL 的链接。这个页面包含六个链接，但有的浏览器可能无法显示包围图像的链接（参见图 6.3.2）

创建其他类型链接的步骤

(1) 输入 <a href="。

(2) 输入 URL。

对于指向万维网上任何文件（包括图像、ZIP 文件、程序、PDF 及其他等）的链接，输入 http://www.site.com/dir/file.ext，其中 www.site.com 是主机名称，dir/file.ext 是目标文件的路径。后者包括了文件目录和文件名（以及扩展名）。

对于电子邮件地址，输入 *mailto:name@domain.com*（不以 http:// 开头），其中 *name@domain.com* 是电子邮件地址（不过，应避免链接到电子邮件地址，参见提示）。

对于电话号码，输入 *tel:+*（不以 http:// 开头）并紧跟着国家代码和电话号码（所有的号码中都不必包含短横线）。号码的长度在每个国家都不一样。例如，美国的国家代码是 1，因此美国的电话号码形如 tel:+18889995555（英国的国家代码为 44，肯尼亚为 254，依次类推）。

(3) 输入 ">。

(4) 输入链接的标签。标签是默认以下划线和其他颜色突出显示的文本，当用户激活它时，会将访问者带到第 (2) 步中引用的 URL。也可以添加一个 img 元素替代文本作为标签（参见图 6.3.1 以及补充材料“将缩略图链接到图像”）。

(5) 输入 。

提 示　如果链接指向的文件是浏览器不知道如何处理的类型（例如 Excel 文件），浏览器将试着打开一个辅助程序来查看这个文件，或试着将它下载到访问者的磁盘上。

提 示　建议不要使用指向电子邮件地址的链接，因为垃圾邮件机器人会从网页上搜集这些地址并向其发送垃圾邮件。最好使用描述性的文字表示电子邮件地址，如“someone at some domain”，不过这样做也并非万无一失。

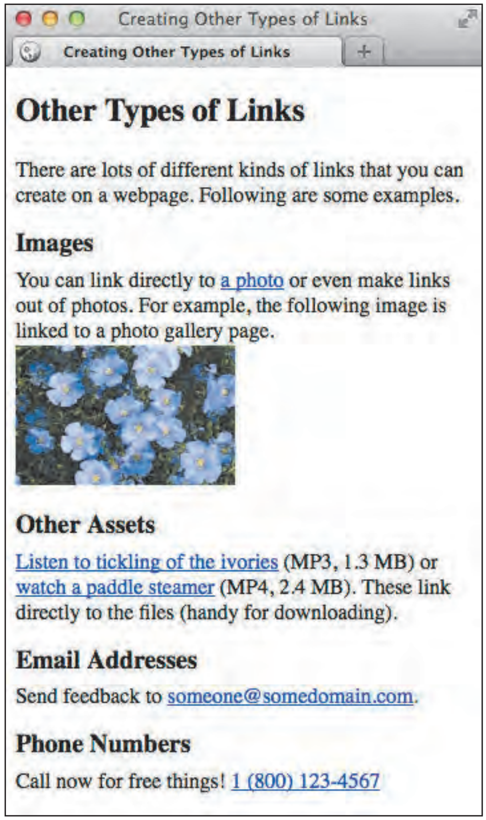


图 6.3.2　无论链接通向哪里，它们在浏览器中的默认样式都是一样的，除非其中包含图像（有的浏览器在图像链接周围添加边框，有的则不会）。注意，我尽量创建与文本主体混在一起的标签，而不是使用“点击此处”（Click me）作为标签

提 示　在理解 tel: 的智能手机里，如果点击这样的链接，手机就会询问用户是否需要拨打该号码。对于一些不是电话的设备（如 iPad），则会询问用户是否需要将该号码添加到通讯录里。此外，有的桌面浏览器在这种情况下会启动 Google Voice 或 Skype，而其他的浏览器则不知道该如何处理这种情况。

提示 尽管可以链接到 PDF 和其他非 HTML 文档（Word、Excel 等），但应尽量避免这样做。相反，应链接到包含有关信息的 HTML 页面。PDF 可能要花更长的时间加载，而且有的浏览器在尝试显示它们的时候会变得很慢。如果 PDF 是唯一的选择，要让用户知道链接指向的是 PDF 而不是另一个 HTML 页面，以免他们感到意外（如果让用户进入耗时的下载，他们是不会领情的）。这条建议对其他非 HTML 文档也适用。显示给用户的信息可以是简单地用括号包围的文件类型和大小，也可以显示一个图标。下面是一个不包含图标的例子：`<a href="q2-sales-report.pdf">Q2 Sales Report` **`(PDF, 725kb)`**`</a>`。还可以在链接中包含 `title` 属性（如 `title="Opens a PDF"`）。

提示 对供访问者下载的大文件和文件组进行压缩是个好办法。例如，一套保存为 PSD 文件的 Photoshop 模板。在网上搜索“ZIP and RAR”，查找创建和打开使用这些流行的压缩格式的文件存档的工具。

提示 如果要“创建指向 iTunes Store、App Store、iBookstore 和 Mac App Store 上内容的链接”（链接如下），可以使用苹果公司的 Link Maker（http://itunes.apple.com/linkmaker）生成 URL 放在 HTML 里。如果你是联盟成员（www.apple.com/itunes/affiliates/），人们通过你的链接购买物品时，苹果公司会为你支付佣金。

将缩略图链接到图像

你肯定访问过显示几个缩略图（图像的微缩版本）的相册页面，其中缩略图链接到大一些的图像。这样就可以一次看到很多图像，让访问者选择要查看全尺寸版本的图像。

实现其基本版本类似于将鲜花图像链接到其他页面的示例代码（参见图 6.3.1）。这些页面中的每一个都包含一个全尺寸照片。（也可以在单独一个动态页面中实现，但这是超出 HTML 能力的高级用法。）

注意不要让任何给定页面上的缩略图数量太多。它们可能很小，但每个缩略图都会生成对 Web 服务器的独立请求，合在一起就会让页面变慢。没有规则规定一个页面放多少缩略图是合适的。这部分取决于页面加载的其他资源的数量和大小，也取决于网站的目标受众。例如，移动设备加载资源通常要慢一些。

因此，如果你有很多缩略图，就要考虑将它们分入多个页面。可以对页面进行测试，再确定最合适的数量。

最后，推荐使用无序列表（`ul`，将在第 15 章讲到）对缩略图列表进行标记。

第 7 章

CSS 构造块

7

本章内容

- 构造样式规则
- 为样式规则添加注释
- 理解继承
- 层叠：当样式发生冲突时
- 属性的值

不得不说，到目前为止，本书演示的绝大多数网页在视觉上都没有什么吸引力。这是因为我们一直将重点放在用 HTML 定义内容上，而不是放在用 CSS（层叠样式表）定义样式上。不过，这一切行将改变。在接下来的几章里，你将学会如何为文本和背景添加样式，实现多栏布局，建立起适应各种设备（从手机到台式计算机甚至屏幕更大的设备）的布局，等等。在这一章，我们将从基本的 CSS 概念讲起，为后面的内容奠定基础。

样式表不过是一种文本文件，其中包含一个或多个（通过属性和值）决定网页某特定元素如何显示的规则。CSS 里有控制基本格式的属性（如 `font-size` 和 `color`），有控制布局的属性（如 `position` 和 `float`），还有决定访问者打印时在哪里换页的打印控制元素。CSS 还有很多控制项目显示或消失的动态属性，可以用于创建下拉列表和其他交互性组件。

CSS2 是新旧浏览器支持最为广泛的版本，因此本书将大量讨论这一版本的内容。CSS3 目前还没有成为规范，它以 CSS2 为基础，提供了大量设计人员和开发人员长期期待的功能。这些功能包括圆角、阴影效果、文字阴影、自定义字体、旋转文本、半透明背景颜色、多图像背景、渐变以及其他很多功能。值得庆幸的是现代浏览器已经实现了一些 CSS3 的组件（且即将实现更多），因此从现在起你就可以使用它们了。

实际上，CSS 在近几年又添加了很多新的特性，以致于我们无法将所有的特性尽数收入本书。哎呀，连 CSS4 都已经提上日程了。不过，本书仍会介绍一些最为有用的 CSS3 特性，并会提供用以学习本书没有提到的一些特性的资源的链接。

CSS 很棒的一点在于开发人员可以在 HTML 页面之外创建 CSS 文件，再将它应用于网站上所有的页面。这在构建网页之初及随后对其进行修改时都极大地简化了样式设置工作。一段时间后，如果需要重新设计网站，而内容和结构保持不变，就可以在 HTML 不发生变动的情况下，为网页提供一套全新的外观。

为充分利用 CSS 的优势，必须一以贯之地依照 HTML 相关章节的推荐做法标记网页。

7.1 构造样式规则

样式表中包含了定义网页外观的规则。

样式表中的每条规则都有两个主要部分：**选择器**（selector）和**声明块**（declaration block）。选择器决定哪些元素受到影响；声明块由一个或多个属性 – 值对（每个属性 – 值对构成一条**声明**，declaration）组成，它们指定应该做什么（参见图 7.1.1 ～图 7.1.4）。

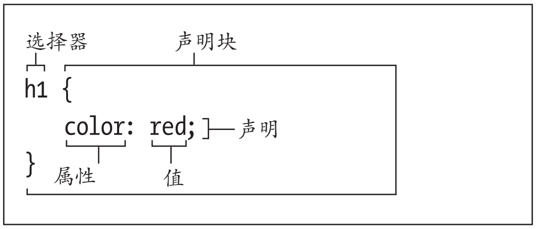


图 7.1.1　样式规则由选择器（表示将对哪些元素进行格式化）和声明块（描述要执行的格式化）组成。声明块内的每条声明都是一个由冒号隔开、以分号结尾的属性 – 值对。声明块以前花括号开始，以后花括号结束

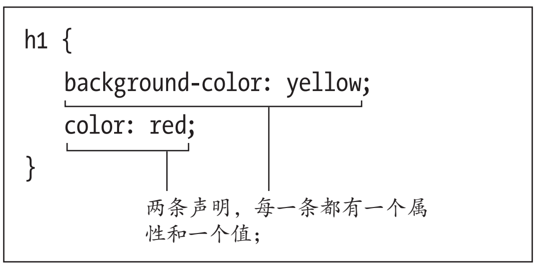


图 7.1.2　声明的顺序并不重要，除非对相同的属性定义了两次。在这个例子中，`color: red` 也可以放在 `background-color: yellow` 前面，效果是一样的。注意额外的空格和缩进（可选，但推荐包含）提高了可读性

```
…
<body>
    <h1>Hey, I've got style!</h1>
</body>
</html>
```

图 7.1.3　如果将图 7.1.2 中的样式规则应用到下面这一小段 HTML 文档中……

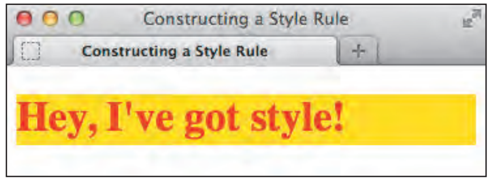


图 7.1.4　标题从浏览器默认的白色背景、黑色文字样式变为黄色背景、红色文字（另见彩插）

构造样式规则的步骤

(1) 输入 *selector*，这里的 *selector* 表示希望进行格式化的元素。第 9 章将讲解如何创建各种类型的选择器。

(2) 输入 {（前花括号）开始声明块。

(3) 输入 *property:value;*，其中 *property* 是 CSS 属性的名称，描述要应用哪种格式；*value* 是该属性允许的选项之一。本书从第 10 章开始讲解 CSS 属性和值。

(4) 根据需要，重复第 (3) 步。通常一行输入一个 `property: value`（一条声明），如图 7.1.2 所示的那样，但这并非强制要求。

(5) 输入 }，结束声明块和样式规则。

提 示　在样式规则中可以添加额外的空格、制表符或回车，从而提高样式表的可读性（参见图 7.1.2）。示例中的格式或许是编码人员中最为常见的一种格式。

提 示　从技术上讲，可以省略样式规则中最后一条声明后面的分号，不过最好加上，这是一种好习惯。

7.2　为样式规则添加注释

在 CSS 中添加注释是个好主意，这样就可以标注样式表的主要区域，或者只是针对某条规则或声明进行解释。注释不仅对开发人员有用，对阅读代码的其他人也有好处。

即便你是网站的唯一开发人员，当你过几个月再看自己写的代码时，会庆幸留下了这些注释。

为样式规则添加注释

(1) 在样式表中，输入 /* 开始注释。

(2) 输入注释。

(3) 输入 */ 结束注释。

提 示 注释可以包含回车，因此可以跨越多行。类似地，/* 和 */ 既可以单独成行，也可以跟文本同处一行，图 7.2.1 中这两种例子都有。另外，注释还可以放到声明块内部，或者样式规则后面，参见图 7.2.2。

```
/*
This is a CSS comment. It can be one line
→long or span several lines. This one
→is much longer than most. Regardless, a
→CSS comment never displays in the
→browser with your site's HTML content.

Of course, you wouldn't really write a
→silly comment like this that merely
→talks about comments. The next comment
→is more in line with a comment's
→typical use.
*/

/* Set default rendering of certain HTML5
→elements for older browsers. */
article,
aside,
figcaption,
figure,
footer,
header,
main,
nav,
section {
    display: block;
}
```

图 7.2.1　注释可长可短，但它们通常较短。可以根据你的想法，用它们描述样式规则或一组相关的规则的目的。注释做得好可以让样式表更易于维护

```
.byline {
    color: green;
    font-size: .875em;
    text-shadow: 2px 1px 5px orange; /* IE9
    →and earlier don't support */
} /* You can put comments here, too! */
```

图 7.2.2　我们也可以在声明块内部或者样式规则后面插入注释

提 示 不能将一个注释嵌套在另一个注释内部，下面的做法是不正确的：

/* 做法错误，因为 /* 这条注释 */ 嵌套在外面这条注释内部 */。

提 示 注释是很有用的组织工具。样式表很快就会变得很长，因此，组织好样式表对于保持 CSS 易于维护至关重要。通常，将相关的规则放在一起，形成分组，并在每组前面放置一段描述性的注释，参见图 7.2.3。

```
/* GLOBAL NAVIGATION
---------------------------------- */
... rules for global nav ...


/* MAIN CONTENT
---------------------------------- */
... rules for main content ...


/* SIGN-UP FORM
---------------------------------- */
... rules for sign-up form ...


/* PAGE FOOTER
---------------------------------- */
... rules for page footer ...
```

图 7.2.3　对样式表中的主要区域添加注释，就可以保持样式表井然有序。下面的格式（使用大写字母和一条下划线）可以很清楚地标识分组的开始位置

7

提示　无论采用哪种注释格式，推荐你定下一种并坚持使用，特别是在团队协作的情况下。

提示　可以将注释放在样式规则内部（参见图 7.2.4）或者周围（参见图 7.2.5），从而对浏览器隐藏样式规则。这是对样式表进行测试的一种好方法，不必永久性地删除注释部分，直到你认为可以删除它们。这也是很有用的调试工具，可以将你认为可能引起问题的地方“注释掉”，再在浏览器中刷新页面，查看问题是不是解决了。

```
img {
    border: 4px solid red;
    /* margin-right: 12px; */
}
```

图 7.2.4　可以将你不希望对页面产生影响的声明“注释掉”。这里，所有的图像都会拥有一条 4 像素宽的红色实线边框，但不会有右侧外边距，因为 margin-right: 12px; 位于一条注释里面

```
p {
    line-height: 1.2;
}

/*
.byline {
    color: black;
    font-size: .875em;
    text-shadow: 2px 1px 5px orange;
}

img {
    border: 4px solid red;
    margin-right: 12px;
}
*/
```

图 7.2.5　注释也可以包围整条规则或者多条规则。在下面这条样式规则中，只应用了段落的 line-height，.byline 和 img 规则被注释掉了。注意，我删除了这两个样式原来的注释（在图 7.2.2 和图 7.2.4 中显示了），这样再在它们周围添加注释就不会导致错误了

提示　出于演示的目的，这些例子都使用了很多注释，但不要因此认为要对所有内容添加注释。如果样式表里的注释过多，反而会难以阅读。要根据需要混合使用组织性注释和描述性注释。要针对你和团队里的其他成员，找到一个平衡点。

7.3　理解继承

继承（inheritance）是 CSS 里一个很重要的概念。看看图 7.3.1 所示的网页。浏览器会将这份 HTML 理解为图 7.3.2 所示的文档树。文档树有助于你理解 CSS。原因是，当你为某个元素应用 CSS 属性时，这些属性不仅会影响该元素，还会影响其下的分支元素。也就是说，这些下层的元素继承了其祖先元素的属性。只不过，它们继承的可不是冷冰冰的钞票，而是颜色、字体大小这样的东西。

```
...
<body>
<div>
    <h1>The Ephemeral Blue Flax</h1>

    <img src="img/blueflax.jpg" width="300"
    → height="175" alt="Blue Flax (Linum lewisii)"/>

    <p>I am continually <em>amazed</em>
    → at the beautiful, delicate Blue Flax
    → that somehow took hold in my garden.
    → They are awash in color every morning,
    → yet not a single flower remains
    → by the afternoon. They are the very
    → definition of ephemeral.</p>

    <p><small>&copy; Blue Flax Society.
    → </small></p>
</div>
</body>
</html>
```

图 7.3.1　所有的内容元素都是 body 元素的后代。在这个例子中，用一个 div 包住了所有的内容。进一步，em 和 small 元素都包含在一个 p 元素里，因此它们是 p 的后代（同时也是 div 和 body 的后代），如图 7.3.2 所示

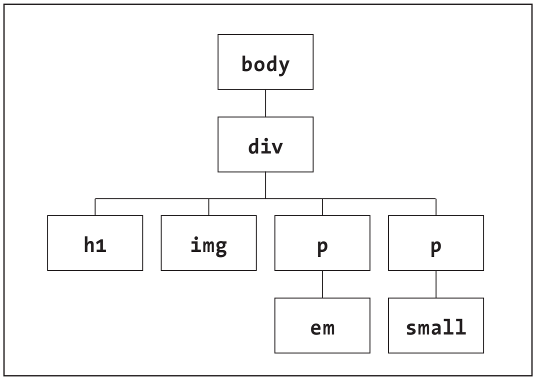


图 7.3.2 将 HTML 页面转换为树形结构后，很容易看出哪些元素是哪些元素的后代。任何一个直接包含在另一个元素中的元素（例如在图 7.3.1 中，img 元素由 div 直接包含），都是其父元素的分支

```
body {
    font-family: Verdana, Geneva,
    → sans-serif;
}
div {
    border: 1px solid #000;
    overflow: hidden;
    padding: 0 1em .25em;
}
p {
    color: #36c; /* a blue color */
    font-weight: bold;
}
img {
    float: left; /* makes text wrap it */
    margin-right: 1em;
}
```

图 7.3.3 这段应用于图 7.3.1 中 HTML 的样式表在为页面设置样式时体现了继承的特性。如果现在不理解其中的细节没有关系，只需注意这里只有 body、div 和 p 元素的样式规则，而没有 h1、em 和 small 元素的规则

图 7.3.1、图 7.3.3 和图 7.3.4 演示了继承的实际效果。例如，font-family 会被继承，而 border 和 padding 则不会继承。这也是为什么页面上的文字使用 Verdana 字体，而只有 div 拥有边框（黑色的细线方框）和内边距（边框和 div 中内容之间的空间），它的后代则没有这些样式，如图 7.3.4 所示。此外，color 和 font-weight 属性也是继承的，因此 em 和 small 元素里的文本也像其他段落文本一样显示为蓝色粗体文字，而不是浏览器默认的黑色非粗体文字。补充材料“哪些属性会被继承”列出了其他一些会被继承的属性。

可见，继承可以简化样式表。设想你要为页面中的每个元素单独设置字体，多么恐怖！在编写 CSS 时，要牢记这一规则，并有效地利用继承。

此外，对大多数属性来说，还可以使用 inherit 值强制进行继承（参见 7.5 节）。

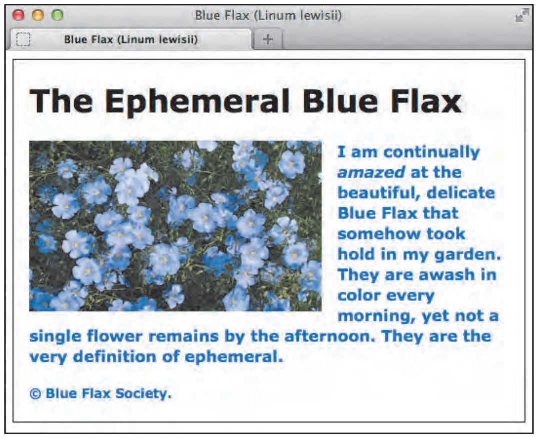


图 7.3.4 这里展示了继承的实际效果（以及未继承的效果）。h1 从 body 中继承了 Verdana 字体样式（由 font-family 设置）。它在图 7.3.3 中没有自己的样式，因此它还会根据浏览器的默认样式进行显示，即显示为黑色粗体的标题样式。类似地，em 和 small 元素也没有指定具体的样式规则，但它们也从 body 继承了 Verdana 字体样式，从 p 继承了 font-weight 和 color 的样式。用 small 标记的文字（法律声明）比周围的文字小一些，这也是浏览器的默认样式。最后，包含所有内容的 div 有一个细线边框和内边距，但其中的其他元素没有这样的样式，因此这些属性不会被继承

哪些属性会被继承

以下是会被继承的 CSS 属性，我们按照类型对其进行了分组。这些属性中的大多数都将在本书后续章节进行讲解，不过你可以根据它们的名称猜出其作用。

◎ 文本

- color（颜色，a 元素除外）
- direction（方向）
- font（字体）
- font-family（字体系列）
- font-size（字体大小）
- font-style（用于设置斜体）
- font-variant（用于设置小型大写字母）
- font-weight（用于设置粗体）
- letter-spacing（字母间距）
- line-height（行高）
- text-align（用于设置对齐方式）
- text-indent（用于设置首行缩进）
- text-transform（用于修改大小写）
- visibility（可见性）
- white-space（用于指定如何处理空格）
- word-spacing（字间距）

◎ 列表

- list-style（列表样式）
- list-style-image（用于为列表指定定制的标记）
- list-style-position（用于确定列表标记的位置）
- list-style-type（用于设置列表的标记）

◎ 表格

- border-collapse（用于控制表格相邻单元格的边框是否合并为单一边框）
- border-spacing（用于指定表格边框之间的空隙大小）
- caption-side（用于设置表格标题的位置）
- empty-cells（用于设置是否显示表格中的空单元格）

◎ 页面设置（对于印刷物）

- orphans（用于设置当元素内部发生分页时在页面底部需要保留的最少行数）
- page-break-inside（用于设置元素内部的分页方式）
- widows（用于设置当元素内部发生分页时在页面顶部需要保留的最少行数）

◎ 其他

- ❏ cursor（鼠标指针）
- ❏ quotes（用于指定引号样式）

7.4 层叠：当规则发生冲突时

样式的来源很多。正如你在第 1 章中学到的，每个浏览器都有其默认样式。不过，你可以用自己的样式覆盖它们或对它们进行补充。应用样式有三种方式：从一个或多个外部文件导入（推荐），如图 7.4.1 中的代码所示，插入到 HTML 文档的顶部，或直接应用于代码中特定的 HTML 元素上（不过，应该尽可能地避免这样做）。下一章将具体讲解三种方式。

你可能会问，对某一给定元素应用多条样式规则时，会发生什么情况？比如一条规则定义了一个元素的颜色，而另一条规则定义了它的宽度，这两条规则会有效结合，一起应用到元素上。

```
p {
    color: red;
}

.example {
    color: blue;
}

.example.example-2 {
    color: magenta;
    /* 会被下一条规则覆盖 */
}

.example.example-2 {
    color: green;
}
```

图 7.4.1 在这个例子中，有四条特殊性不尽相同的样式规则。第一条影响所有 p 元素，第二条仅影响图 7.4.2 的 HTML 中类名为 example 的元素，第三条和第四条则影响类名同时包含 example 和 example2 的元素。这些规则的顺序并不重要（除了最后两条，因为它们的选择器完全相同）

```
...
    <link rel="stylesheet" href="style.css" />
</head>
<body>

<p>Here's a generic <code>p</code> element. It will be red.</p>

<p class="example">Here's a <code>p</code> element with a <code>class</code> of <code>example
➝ </code>. There are two rules that could apply, but since the <code>.example</code> selector is
➝ more specific, this paragraph will be blue.</p>

<p class="example example-2">Here's a <code>p</code> element with two classes: <code>example
➝ </code> and <code>example-2</code>. There are four rules that could apply to this paragraph. The
➝ first two are overruled by the more specific last two. However, because the last two have the
➝ same selector, the order breaks the tie between them: the one that appears later wins, and thus
➝ this paragraph will be green instead of magenta.</p>

</body>
</html>
```

图 7.4.2 这里有三个段落：一个普通的，一个带有一个类名，一个带有两个类名。注意，如果一个元素需要包含一个以上的类名，在 HTML 中应使用空格分隔不同的类名，但在 CSS 中不能像这样定位元素。在 CSS 中，类名的前面使用点号（.）。图 7.4.1 中的样式表文件名为 style.css，第二行中的 link 标签会将该文件加载到页面中，这些内容将在下一章进行讲解

然而，有时候多条规则会定义元素的**同一个**属性（参见图 7.4.1），这时该怎么办呢？CSS 用层叠的原则来考虑样式声明，从而判断相互冲突的规则中哪个规则应该起作用。首先，你编写的样式如果与浏览器的默认样式冲突，均以你编写的样式为准。在此基础上，CSS 用层叠的原则来考虑**特殊性**（specificity）、**顺序**（order）和**重要性**（importance），从而判断相互冲突的规则中哪个规则应该起作用。不要受这些术语的影响，你只要去试，就能明白 CSS 决定该应用哪些样式以及何时应用这些样式的方式。

接下来，我们详细介绍特殊性、顺序和重要性。

1. 特殊性

特殊性规则指定选择器的具体程度。选择器越特殊，规则就越强。遇到冲突时，优先应用特殊性强的规则，如图 7.4.3 和图 7.4.4 所示。讲得通，对吧？

我们不限于使用示例中显示的基本选择器来定义样式。第 9 章会涉及特殊性权重各不相同的其他类型的选择器。我们认为 `id` 选择器是最特殊的（因为它们在一个文件中必须是唯一的），如图 7.4.4 所示。另一方面，除了浏览器默认样式，我们认为继承的样式是最一般的，可以被任何其他规则覆盖。

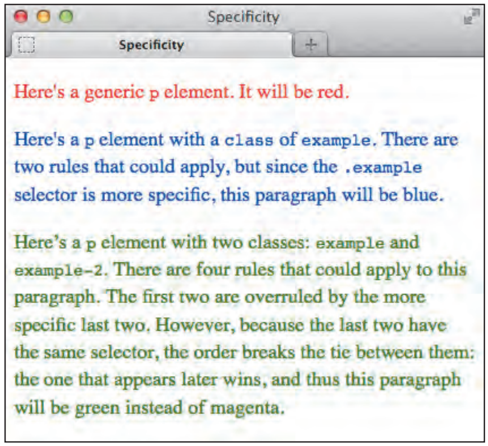


图 7.4.3　第二个段落是蓝色的，因为它的类名包括 example。对于第三个段落，由于第三条和第四条规则的特殊性相同，因此它们的顺序变为决定性因素——第四条规则后出现，因此它起作用

选择器	对应的 HTML
`p { ... }`	`<p>...</p>`
`.someClass { ... }`	`<p class="someClass">...</p>`
`.someClass.someOtherClass { ... }`	`<p class="someClass someOtherClass">...</p>`

`#someID { ... }`	`<p id="someID">...</p>`
	`<p id="someID" class="someClass">...</p>`
	`<p id="someID" class="someOtherClass">...</p>`
	`<p id="someID" class="someClass someOtherClass">...</p>`

图 7.4.4　这里列出了几个选择器，按照特殊性由低向高排列。其中，特殊性最低的是元素名本身（最上面），特殊性最高的是 ID（最下面）。图 7.4.1 ～图 7.4.3 反映了这一行为规则

ID 选择器将在第 9 章进行讲解（参见 9.3 节），不过现在要提一句，建议在样式表中多使用类选择器，避免使用 ID 选择器。使用 ID 选择器通常会矫枉过正，降低了灵活性。

2. 顺序

有时候，特殊性还不足以判断在相互冲突的规则中应该优先应用哪一个。在这种情况下，规则的**顺序**就可以起到决定作用：晚出现的优先级高（参见图 7.4.1、图 7.4.2 和图 7.4.3）。例如，直接应用在 HTML 元素上的规则（不推荐使用这种做法）被认为比外部样式表中或插在 HTML 文档顶部的特殊性相同的规则出现得更晚（因此优先级更高）。

3. 重要性

如果这还不够，可以声明一条特殊的规则覆盖整个系统中的规则，这条规则的重要程度要比其他所有规则高。也可以在某条声明的末尾加上 `!important`，比如 `p{ color: orange !important; }`（除非是在特殊情况下，否则不推荐使用这种方法）。

相关细节参见 8.5 节。

4. 小结

你编写的样式会覆盖浏览器的默认样式。当两个或两个以上的样式发生冲突时，会应用特殊性高的样式声明，不管它位于样式表中的哪个位置。如果两个或两个以上的规则拥有相同的特殊性，则使用后出现的规则，除非其中某条规则标记了 `!important`。

如果某元素没有直接指定某条规则，则使用继承的值（如果有的话）。

如果这些内容看起来有些令人困惑，现在也不必担心。当你试着编写 CSS，开始使用一些选择器后，你会发现，在大多数情况下，层叠的运行规则跟你设想的是一样的。

提示　如果你需要复习 HTML 中的类和 ID 名称，参见 3.14 节。

提示　如果你对浏览器计算特殊性权重的具体细节感兴趣，可以查看 CSS 规范的第 9 部分（www.w3.org/TR/selectors/#specificity）。注意，在学习本书第 9 章之前，你可能会对规范中的内容感到陌生。其实，编写 CSS 不必死记硬背公式化的内容。几乎没有多少代码编写人员能背下这些特殊性规则，即便那些具有多年经验的程序员也是如此。他们（以及你）不必死记硬背的原因是 CSS 特殊性规则实际上相当直观。如果一条样式并未按照你设想的方式呈现，很有可能是其特殊性不够高，这样，你就可以从编码的过程中得到经验。

提示　先前提到，样式表可能来自于浏览器和代码编写人员。实际上还有第三种来源，即访问者（尽管这种来源并不常见）。有的浏览器允许用户创建自己的样式表并将其应用到他们所访问的页面（包括代码编写人员的页面），从而根据自己的喜好定制体验。例如，一个有视觉障碍的用户可能希望文本和背景颜色的对比度更大一些。实际上，没有多少用户知道浏览器的这一特性，使用它的用户就更少了。这些样式确实有可能影响层叠。它们的优先级高于浏览器的默认样式，但低于你为网站创建的 CSS（除非用户的样式表声明中标记了 `!important`）。提示这一点主要是为了让你了解这种用户行为，不必因此对设置网页样式的方法作出任何改变。

7

7.5　属性的值

每个 CSS 属性对于它可以接受哪些值都有不同的规定。有的属性只能接受预定义的值。有的属性接受数字、整数、相对值、百分数、URL 或者颜色。有的属性可以接受多种类型的值。每个属性可接受的值将在描述该属性的章节（大多在第 10 章、第 11 章和第 14 章）列出来，不过这里将讨论基本体系。

1. inherit

对于任何属性，如果希望显式地指出该属性的值与对应元素的父元素对该属性设定的值相同，就可以使用 `inherit` 值。例如，假设有一个 `article` 元素，其中包含几个段落。`article` 元素设置了一个边框。边框通常不会被继承，因此 `p { border: inherit; }` 这条规则可以让这些段落获得相同的边框样式。（在 IE8 之前的 Internet Explorer 版本中，大部分属性都不支持 `inherit` 值，不过你大概也不需要支持那些大多数网站都不再支持的旧浏览器。）

2. 预定义的值

大多数 CSS 属性都有一些可供使用的预定义值。例如，`float` 属性可被设为 `left`、`right` 或 `none`。与 HTML 不同，不需要（也**不能**）将预定义的值放在引号里，参见图 7.5.1。实际上，大多数 CSS 值，无论是否为预定义的值，都不需要加引号。（也有一些例外，如超过一个单词的 `font-family` 名称，以及要生成的内容。）

```
        预设值
border: none;
```

图 7.5.1　很多 CSS 属性只接受预定义的值。要准确地输入这些值，并且不要加上引号

3. 长度和百分数

很多 CSS 属性的值是长度。所有长度都必须包含数字和单位，并且它们之间没有空格。例如 `3em`、`10px`，参见图 7.5.2。唯一的例外是 `0`，它可以带单位也可以不带。效果是一样的，因此 0 一般不带单位。

```
             长度
font-size: .875em;
```

图 7.5.2　必须显式指出长度的单位。在单位和数值之间应该没有空格

有的长度是相对于其他值的。一个 em 的长度大约与对应元素的字号相等。例如，对元素设置 `margin-left: 2em` 就代表将元素的左外边距设为元素字号的两倍。（当 em 用于设置元素的 `font-size` 属性本身时，它的值继承自对应元素的父元素的字号，第 10 章会讨论这一点。）em 的这种相对性对如今的网站建设工作来说是至关重要的，尤其是对那些需要适应不同屏幕尺寸的网站来说（这种做法被称为**响应式 Web 设计**，将在第 12 章介绍）。此外，rem 单位是相对于 `html` 元素的字体大小的，参见 10.6 节。

如果用过 Adobe Photoshop 这样的软件，你一定很熟悉像素的概念。跟 em 不同，**像素**（px）并不是相对于其他样式规则的。因此以 px 为单位的值不会像以 em 为单位的值那样受 `font-size` 设置的影响。不过，在如今设备种类繁多的情况下，已经很难再准确度量一个像素的实际大小了。某种设备上一个像素的大小不一定与另一种设备上的完全相等。（参见 Peter-Paul Koch 对此问题的详细说明，www.quirksmode.org/blog/archives/2010/04/a_pixel_is_not.html。警告：有些内容可能不好懂。）有时候我们需要使用 px，只是没那么

频繁。

还有一些无需说明的绝对单位，如磅（pt），应该在为打印准备的样式表中保留这个单位。一般来说，应该只在输出尺寸已知的情况下使用绝对长度（如在打印的页面中使用 pt）。

百分数（如 65%）的工作方式很像 em，它们都是相对于其他值的值，参见图 7.5.3。出于这个原因，百分数是创建我们很快就要学习的响应式网站的一个强大工具。

百分数

```
width: 80%;
```

图 7.5.3 百分数通常是相对于父元素的。因此，在这个例子中，字号将被设为父元素字号的 80%

在上述单位中，最常使用的是 em、百分数和像素，rem 的使用情况越来越多。你的样式表可以自由组合使用上述单位，即使是在同一条样式规则中，也可以组合使用不同的单位。还有其他一些单位，不过鲜有使用。

4. 纯数字

只有极少数的 CSS 属性接受不带单位的数字，如 3、0.65。其中最常见的就是 line-height（参见图 7.5.4）、z-index 和 opacity（分别参见 10.7 节、11.15 节和 14.8 节）。

数字

```
line-height: 1.5;
```

图 7.5.4 不要将数字、整数和长度弄混。数字或整数没有单位（如 px）。这个例子中的值将与字号相乘，得到行高

5. URL

有的 CSS 属性允许开发人员指定另一个文件（尤其是图像）的 URL，background-image 就是这样一个属性。在这种情况下，使用 url(*file.ext*)，其中 *file.ext* 是目标资源的路径和文件名，参见图 7.5.5。注意，规范指出，相对 URL 应该相对于样式表的位置而不是相对于 HTML 文档的位置。

URL

```
background: url(bg-pattern.png);
```

图 7.5.5 CSS 属性中的 URL 无需使用引号包围

可以在文件名上加上引号，但这不是必需的。此外，在单词 url 和前括号之间不应该有空格。括号和地址之间允许有空格，但这不是必需的（通常也不这样做）。

关于编写 URL 本身的信息，参见 1.7 节。

6. CSS 颜色

我们可以使用预定义颜色关键字或以十六进制（通常称为 hex）、RGB、HSL、RGBA、HSLA 等格式表示的值中为 CSS 属性指定颜色。最后两种格式可以指定具有一定程度 alpha 透明度的颜色。HSL、RGBA 和 HSLA 格式都是在 CSS3 中引入的。

CSS3 指定了 CSS2.1 本来就有的 16 个基本的名称（参见图 7.5.6），另外又增加了 131 个。完整的列表见 www.w3.org/TR/css3-color/#svg-color。

当然，除了几样最简单的名称，没有人记得住这些颜色名。可以使用 Adobe Photoshop 等工具进行取色，但它们并不使用 CSS 颜色名，你获取的是 RGB 或十六进制值。此外，颜色关键字也只是你使用的颜色中很小的一部分，因此在实践中，定义 CSS 颜色更常规的方法是使用十六进制格式（这是目前为止最为常见的方式）或 RGB 格式。当然，

要指定 alpha 透明度的情况除外，这时应该用 RGBA 和 HSLA 格式。

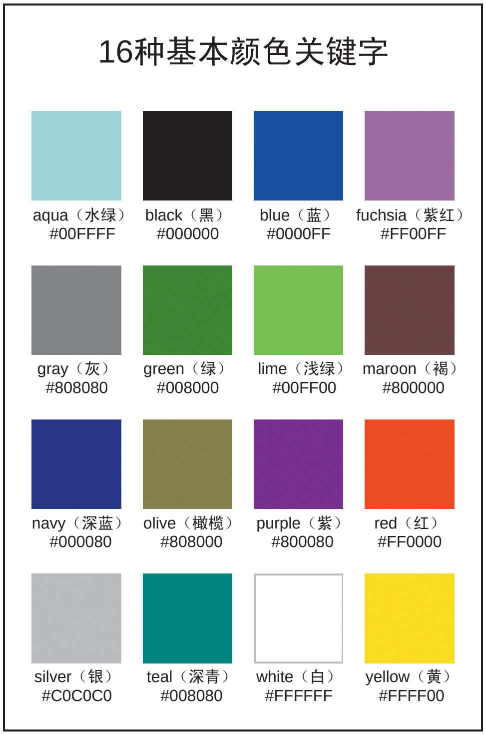


图 7.5.6　16 种基本颜色关键字，以及它们对应的十六进制数字值。CSS3 新增了 131 个关键字，不过通过十六进制、RGB 或 HSL 等格式可以定义多得多的颜色。另见彩插

7. RGB

可以通过指定红、绿、蓝（这也是 RGB 这一名称的由来）的量来构建自己的颜色。可以使用百分数、0 ~ 255 之间的数字来指定这三种颜色的值。例如，如果创建一种深紫色，可以使用 89 份红、127 份蓝、0 份绿。这个颜色可以写做 `rgb(89, 0, 127)`，如图 7.5.7 所示。

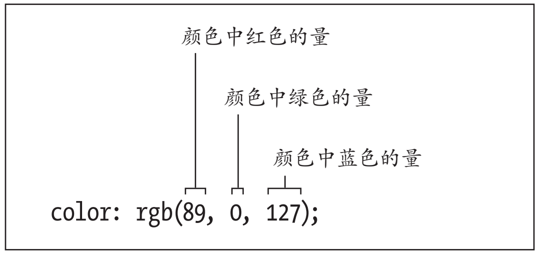


图 7.5.7　表示 CSS 颜色的另一种方式是用 0 ~ 255 之间的数字指示 RGB 数值。首先定义红色，然后是绿色，最后是蓝色

此外，也可以将每个值表示为百分数，不过很少用到这种做法，可能因为 Photoshop 等图像编辑器通常用数字表示 RGB 值。如果你想使用百分数，可以将上面的颜色写作 `rgb(35%, 0%, 50%)`，因为 89 是 255 的 35%，127 是 255 的 50%。

8. 十六进制数

我把最常用的方法放在最后讲解，参见图 7.5.8。将这些数字值转化为十六进制，然后将它们合并到一起，再在前面加一个 #，就像 `#59007F` 这样。你心里可能想：“为什么不用十进制呢？”别慌，类似于 Photoshop 这样的工具在选择颜色时可以显示颜色的 RGB 值，以及对应的十六进制数。此外你也可以上网搜索免费的转换工具（如输入 rgb to hex converter）。

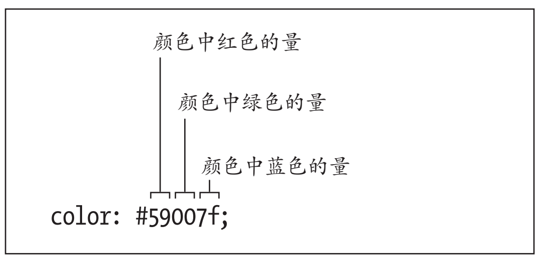


图 7.5.8　CSS 中定义颜色最常用的方式是用十六进制数指定颜色所包含的红、绿、蓝的量

对于 `#59007F`，十六进制的 59、00、7F 分别等于 89、0、127。`7F` 和 `7f` 都是允许的写法（我倾向于使用后一种写法，不过很多开发人员和设计人员都选择前一种）。

如果一个十六进制颜色是由三对重复的数字组成的，如 `#ff3344`，可以缩写为 `#f34`。这种做法也是一种最佳实践，因为没有理由让代码无谓地变长。

9. 更多 CSS3 提供的指定颜色的方式：RGBA、HSLA 和 HSL

CSS3 引入了另一种指定颜色的方式——HSL，以及通过 RGBA 和 HSLA 设置 alpha 透明度的能力。（使用十六进制记法无法指示 alpha 透明度。）

◎ RGBA

RGBA 在 RGB 的基础上加了一个代表 alpha **透明度**（alpha transparency）的 A。alpha 透明度的一种常见使用场景是将其用在对元素设置 `background-color` 或 `background` 的情况（均用于设置背景），参见图 7.5.9 和图 7.5.10，因为 alpha 透明度允许元素下面的任何东西（如图像、其他颜色、文本等）透过来并与元素混合在一起（如图 7.5.11 所示）。也可以对其他基于颜色的属性使用 `alpha` 透明度，如 `color`、`border`、`border-color`、`box-shadow`、`text-shadow` 等。所有现代浏览器都支持 `alpha` 透明度，但 IE9 之前的 Internet Explorer 不支持（如图 7.5.12 所示）。

可以在红、绿、蓝数值后面加上一个用以指定透明度的 0 到 1 之间的小数。语法如下：

```
property: rgba(red, green, blue, alpha
→ transparency);
```

alpha 设置越接近 0，颜色就越透明。如果设为 0，就是完全透明的，就像没有设置任何颜色。类似地，1 表示完全不透明。

为了解释上面的说法，下面给出了一些声明示例：

```
/* 不透明，和 rgb(89, 0, 127); 效果相同 */
background-color: rgba(89,0,127,1);
/* 完全透明 */
background-color: rgba(89,0,127,0);
/* 25% 透明 */
background-color: rgba(89,0,127,0.75);
/* 60% 透明 */
background-color: rgba(89,0,127,0.4);
```

当然，为了让这些记法正常运行，需要将它们放入一个或多个规则中（参见图 7.5.9）。可以看到，尽管 RGB 颜色值是一样的，但由于设置了不同的透明度，颜色在浏览器中显示的样子就不一样了，参见图 7.5.11。

◎ HSL 和 HSLA

HSL 和 HSLA 也是 CSS3 中新增的。HSLA 是除 RGBA 以外为颜色设置 alpha 透明度的另一种方式。用 HSLA 指定 alpha 透明度的方法与 RGBA 是一致的。待会儿再看 HSLA，先看看 HSL 的工作原理。

HSL 代表**色相**（hue）、**饱和度**（saturation）和**亮度**（lightness），其中色相的取值范围为 0 ~ 360，饱和度和亮度的取值均为百分数，范围为 0 ~ 100%，如图 7.5.13 所示，0 和 360 在顶部汇合，意思是 0 和 360 代表同一种颜色：红色。饱和度和亮度设置会直接应用到颜色上（不要将 HSL 和 HSB、HSV 混淆。它们很相似，但不是一回事，CSS 不允许用 HSB 和 HSV 表示颜色）。

```
/* 设置重复页面背景图像和默认文字颜色、字体、
   字号 */
body {
     background: url(../img/bg-pattern.png);

     /* 由于继承原因，这些文字样式会应用到所
        有的文字，除非定义了其他样式覆盖前面
        的样式*/
     color: #ff0; /* 十六进制黄色 */
     font-family: arial, helvetica,
     → sans-serif;
     font-size: 100%;
}

/* 所有段落都会显示纯色背景，除非指定下列类别
   之一。IE9之前的版本不支持透明色，无论如何都
   会显示背景色 */
p {
     background-color: rgb(60,143,0);
     font-size: 1.75em;
     padding: .5em;
}

/* 25% 透明度 */
.test-1 {
     background-color: rgba(60,143,0,.75);
}

/* 60% 透明度 */
.test-2 {
     background-color: rgba(60,143,0,.4);
}
```

图 7.5.9　这份简单的样式表对整个页面应用了一个重复显示的背景图，设置了默认文字颜色，并依据类别的不同对各个段落添加了不同的背景，参见图 7.5.10。现代浏览器显示的结果如图 7.5.11 所示。IE9 之前的版本不支持 RGBA，因此它们会忽略对 .test-1 和 .test-2 的声明，直接显示为 p 应用的 RGB 颜色样式，参见图 7.5.12

在 CSS 中，HSL 语法为（具体参见图 7.5.14）：

```
property: hsl(hue, saturation, lightness);
```

HSLA 的格式为：

```
property: hsla(hue, saturation,
→ lightness, alpha transparency);
```

看一下更新后的样式表（其中省略了部分相同的代码），如图 7.5.15 所示。这张样式表使用 HSL 和 HSLA 定义了与前例中使用 RGB 和 RGBA 定义的同样的绿色。

```
...
<body>

<p class="test-1">This background is 25%
→ transparent.</p>

<p class="test-2">This background is 60%
→ transparent.</p>

<p>This has the same background color as
→ the other two, but doesn't have an alpha
→ transparency setting.</p>

</body>
</html>
```

图 7.5.10　第一个段落有 test-1 的类名，因此根据样式表，它有 25% 透明的背景色。第二个段落有 test-2 的类名，因此有 60% 透明的背景色。最后一个段落没有任何类名，因此它的背景色是纯色的，由图 7.5.9 中对所有 p 元素定义的样式（第二条规则）所指定

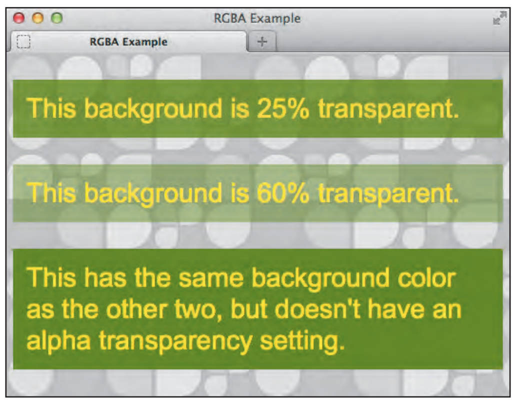


图 7.5.11　我们可以看到页面背景图透过了前两个段落的背景，但没有透过最后一个段落的背景。这三个段落的背景色都是一样的，但由于 alpha 透明度不同，它们看起来就像是深浅不同的三种绿色(参见图 7.5.9 中 body 规则的 CSS 注释，了解如何实现这里的文本样式）

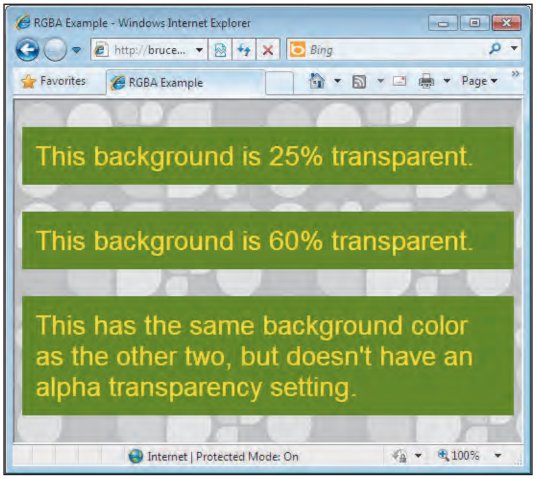


图 7.5.12 IE 直到 IE9 才支持 alpha 透明度（RGBA 或 HSLA）。这是 IE8 中的截图。IE8 忽略了样式表中的 GRBA 声明，因为它并不理解这些规则。因此所有的段落都是纯色背景（由对 p 元素进行定义的规则指定）。如果没有这条规则，则段落不会有任何背景色（段落中的文字现在看起来已经不正确了，这是因为现在没有透明效果）



图 7.5.13 HSL 色相值沿顺时针改变。最右边的点为 90，最底部的点为 180，最左边的点为 270，0 和 360 在顶端重合。关于标准颜色及色相值参见补充材料“如何构想 HSL”

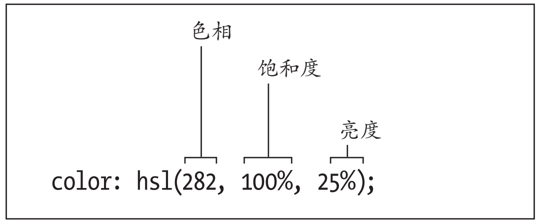


图 7.5.14 对 HSL 格式的分解

```
body {
    … 与前例相同的样式 …
}

/* 所有段落都会显示纯色背景，除非指定下列类
   别之一。IE9之前的版本不支持透明色，只显示
   十六进制颜色 */
p {
    background-color: #3c8f00; /* for IE
    ➝ before IE9 */
    background-color: hsl(95,100%,28%);
    …
}

/* 25% 透明度 */
.test-1 {
    background-color:
    ➝ hsla(95,100%,28%,.75);
}

/* 60% 透明度 */
.test-2 {
    background-color: hsla(95,100%,28%,.4);
}
```

图 7.5.15 同样的样式表，但使用 HSL 和 HSLA 表示。此外，我们还针对旧浏览器为段落定义了备选的 background-color（背景色）。p 中声明的顺序是有要求的。旧版本的 IE 使用第一行，因为它能理解这一行，而现代浏览器对这两行都理解，它会应用第二行的样式，因为它出现在后面

如果你仔细看看这些定义 p 元素样式的规则，你会发现一些有趣的事情。这里使用十六进制数定义了 background-color，紧接着又使用 HSL 格式重新定义了一遍。这样做的原因是 Internet Explorer 的所有版本都理解十六进制值，但 IE9 之前的版本无法理解

HSL 或 HSLA。这些旧的浏览器会忽略 HSL 和 HSLA 样式，就像忽略 RGBA 一样，而其他的浏览器则会应用这些定义。这样做的结果就是新的样式表在现代浏览器中保持了一致的效果（如图 7.5.11 所示），而 IE8 也没有变化（如图 7.5.12 所示）。

并非所有的图像编辑器都可以在对话框中指定 HSL（可以为 Photoshop 安装一个插件）。不过，通过 Brandon Mathis 强大的免费在线工具 HSL Color Picker（hslpicker.com）可以选取颜色，获取其 HSL 值、十六进制数值和 RGB 值。还可以输入这些格式的颜色值，查看颜色的变化。通过在线搜索还可以找到更多其他的颜色工具。

提示 前面已经提到，IE 直到 IE9 才能理解 RGBA、HSL 和 HSLA。可以为这些旧浏览器单独定义备选的纯色（参见图 7.5.9），也可以在同一条规则中提供备选的纯色（参见图 7.5.15）。

提示 如果确实想在旧版本的 IE 中使用 alpha 透明度，也可以找到一些方法。一种方法是使用一个 1 像素 ×1 像素的半透明 PNG 作为重复的背景（参见 10.10 节），在使用 RGBA 或 HSLA 的属性之前使用该 PNG 定义备选背景（类似于图 7.5.15）。可以在 rgbapng.com 网站上创建所需的图像，只是需要注意不要在文件名中使用 .part 的后缀（这是它默认添加的）。另一种方法是使用 IE 专有的 Gradient 滤镜。它的语法很令人崩溃，不过好在有专门创建这些代码的工具，如 Michael Bester 的 RGBA & HSLA CSS Generator for Internet Explorer（http://kimili.com/journal/rgba-hsla-css-generator-for-internet-explorer）。注意，过多地使用滤镜可能会减慢页面在 IE 中的加载速度。

如何构想 HSL

学习 HSL 的逻辑需要花些时间，但是一旦找到感觉，你会发现它比其他的格式更容易使用。Brandon Mathis 在其 HSL Color Picker 网站的”Why?“（为何使用 HSL）部分对这种格式有一段精彩的解释。他写道：

“选择一个 0 到 360 之间的色相，并将饱和度设为 100，亮度设为 50，就会得到这种颜色最纯的形式。降低饱和度，颜色就会向灰色变化。增加亮度，颜色就会向白色变化；减少亮度，颜色就会向黑色变化。”

例如，沿着圆环移动的过程中可以得到下面这些比较重要的颜色：

- 红色为 hsl(**0**,100%,50%);
- 黄色为 hsl(**60**,100%,50%);
- 绿色为 hsl(**120**,100%,50%);
- 青色为 hsl(**180**,100%,50%);
- 蓝色为 hsl(**240**,100%,50%);
- 紫红色为 hsl(**300**,100%,50%);

第 8 章

操作样式表

8

本章内容

- ❑ 创建外部样式表
- ❑ 链接到外部样式表
- ❑ 创建嵌入式样式表
- ❑ 应用内联样式
- ❑ 样式表的层叠和顺序
- ❑ 使用与媒体相关的样式表
- ❑ 借鉴其他人的灵感：CSS

在开始定义样式表之前，要知道如何创建和使用包含这些样式表的文件。在本章中，你将学习如何创建样式表文件，如何将 CSS 应用到多个网页（包括整个网站）、单个页面或单独的 HTML 元素。我们会通过三种方法实现这些应用：外部样式表（首选方法）、嵌入样式表和内联样式（最不可取的方法）。

在随后的几章，我们将学习如何创建样式表中的内容。

8.1 创建外部样式表

外部样式表非常适合给网站上的大多数页面或者所有页面设置一致的外观。可以在一个外部样式表中定义全部样式，然后让网站上的每个页面加载这个外部样式表，从而确保每个页面都有相同的设置。尽管后面你将学到嵌入样式表和内联样式表，但从外部样式表为页面添加样式才是最佳实践，推荐使用这种方法（允许偶尔的例外）。

创建外部样式表的步骤

(1) 在你选择的文本编辑器（参见第一条提示）中创建一个新文档。大多数人使用同一个编辑器创建 HTML 和 CSS 文档。

(2) 为网页定义样式规则，同时，根据需要在 CSS 中添加注释（参见 7.1 节和 7.2 节），如图 8.1.1 所示。

(3) 将文档以纯文本格式保存在希望放置的目录中。尽管任何名称都是允许的，但应该为该文件添加 .css 的扩展名，表明这是一个层叠样式表，如图 8.1.2 所示。

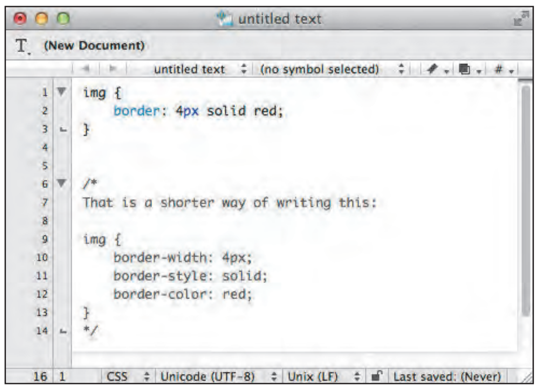


图 8.1.1　使用任何你喜欢的文本编辑器创建样式表。这是 Mac 上的 TextWrangler。我在文档底部包含了一条 CSS 注释，以表明 `border` 属性的工作原理，这跟 8.5 节中的内容相关。这里绝对不是建议你为样式表中的每一条规则添加注释

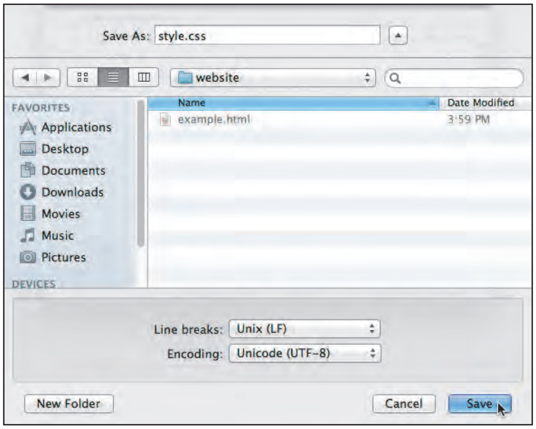


图 8.1.2　一定要将 CSS 文件保存为以 .css 为扩展名、用 UTF-8 编码的文件。有的编辑器需要用户指定将文件保存为纯文本格式（编辑器可能称其为 Text Document（文本文档）、Plain Text（纯文本）、ASCII 等），但 TextWrangler 并不需要。Notepad 会要求这样做，更多信息参见 2.3 节。注意这里将 style.css 与将会用到它的 HTML 页面（example.html）放在同一个目录。这样做可以让这个例子变得相当简单，但在实践中，最好将样式表文件放在单独的目录，从而让网站具有良好的组织结构

在下一节，你将学到如何将样式表加载到 HTML 页面里，从而让样式规则应用到页面上。

提 示　关于可用于编写 CSS 的文本编辑器的信息，参见 2.2 节。

提 示　如果你现在对 CSS 本身还不太理解，不必担心。在接下来的几章，你将学会如何创建这样的样式规则。不过通常都可以猜出效果，例如，本节例子中规则的意思是“为页面上所有 img 元素添加 4 像素宽的红色实线边框”。

提 示　可以以任何名称为样式表文件命名。因此，如果网站只有一个样式表文件，通常会命名为 style.css 或 styles.css。更大的网站则通常会拥有多个样式表文件。在那种情况下，base.css、global.css 和 main.css 是常见的样式表名称，它们通常包含应用于网站大多数页面的样式规则。网站制作者通常创建一些为某些区块所特有的附加 CSS 文件（它们都会被加载到相关页面），作为对基本样式的补充。例如，对于一个商业网站，products.css 包含的可能是为产品相关页面准备的样式规则。无论选择什么文件名，一定不要包含空格。

提 示　外部样式表要么是通过链接引用的(参见 8.2 节），要么是导入的（通过 `@import`），不过不推荐导入。`@import` 指令会影响页面的下载速度和呈现速度，在 Internet Explorer 中影响更为明显。Web 性能专家 Steve Souders 对此问题进行了讨论，参见 www.stevesouders.com/blog/2009/04/09/dont-use-import/。如果读者对 `@import` 好奇，请查阅 http://reference.sitepoint.com/css/at-import 了解基础知识。

提 示　CSS 注释（参见图 8.1.1）既不会影响页面的呈现，也不会显示在网页上，但如果访问者查看源代码，就能看到它们（参见本章后面的 8.7 节）。

8.2　链接到外部样式表

创建了样式表（参见图 8.2.1）之后，需要将它加载到 HTML 页面中去，从而为内容应用这些样式规则。要做到这一点，最好的方式是链接到样式表（参见图 8.2.2）。

```
img {
    border: 4px solid red;
}
```

图 8.2.1 这是本章前面创建的外部样式表 style.css，简洁起见，省略了 CSS 注释

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>El Palau de la Música</title>
    <link rel="stylesheet"
    → href="style.css" />
</head>
<body>
<article>
    <h1>El Palau de la Música</h1>

    <img src="img/palau.jpg" width="250"
    → height="163" alt="El Palau de la Música" />

    <img src="img/tickets.jpg" width="87"
    → height="163" alt="The Ticket Window" />

    <p>I love the <span lang="es">Palau de la
    → Música</span>. It is ornate and gaudy
    → and everything that was wonderful
    → about modernism. It's also the home
    → of the <span lang="es">Orfeó Català
    → </span>, where I learned the benefits
    → of Moscatell.</p>
</article>
</body>
</html>
```

图 8.2.2 link 元素位于 HTML 文档的 head 部分。页面可以包含一个以上的 link 元素，但使用它的次数最好尽可能地少，让页面得以更快地加载

链接到外部样式表的步骤

(1) 在每个希望使用样式表的 HTML 页面的 head 部分，输入 <link rel="stylesheet"。

(2) 输入一个空格，然后输入 href="*url.css*"，其中 *url.css* 是CSS样式表的位置和名称。

(3) 输入一个空格和 />。（也可以不输入空格，直接输入 >。这两种形式都是 HTML5 所允许的，它们的效果也完全一样。）

提 示 对外部样式表进行修改时，所有引用它的页面也会自动更新（参见图 8.2.3 和图 8.2.4）。这是外部样式表的神奇力量！



图 8.2.3 将这个样式规则（4 像素宽的红色实线边框）应用于每个 img 元素

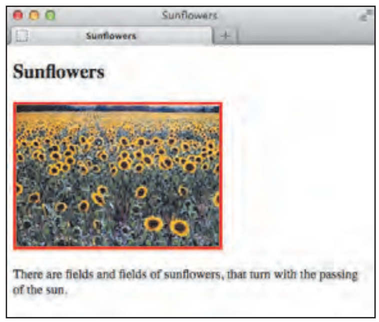


图 8.2.4 其他文档也可以链接到同一个样式表，并拥有同样的样式

8

提 示 外部样式表的另一个好处是，一旦浏览器在某个页面加载了它，在随后浏览引用它的页面时，通常无需再向 Web 服务器请求该文件。浏览器会将它保存到**缓存**里，也就是保存到用户的计算机里，并使用这个版本的文件。这样做可以加快对页面的加载。不过，不必担心。如果随后对样式表作了修改，再将它传到 Web 服务器，浏览器就会下载更新后的文件，而不是使用缓存的文件（从技术上讲也有例外，但通常不会遇到这种情况）。

提示 出于简化的目的，这个例子中的链接假定 HTML 页面与 style.css 位于同一个目录（参见图 8.2.2）。不过，实践中最好将样式表组织在子文件夹里，而不是与 HTML 页面混在一起。常见的样式表文件夹名称包括 css、styles 等，你也可以按照自己的意愿对其进行命名，只要你认为该名称放在链接的 href 值里是合适的。例如，如果 style.css 位于名为 css 的文件夹里，而 HTML 位于该文件夹的上一级（参见图 8.2.5），那么 link 元素就应该写作图 8.2.6 中那样。

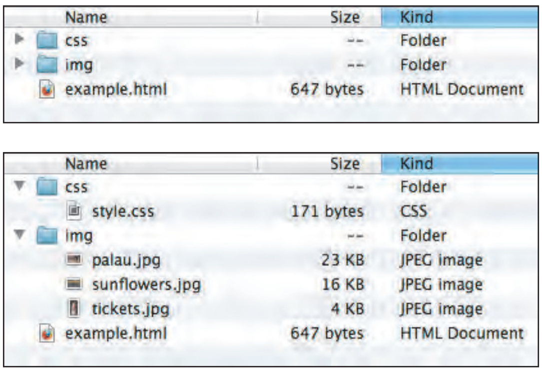


图 8.2.5　这些截图显示了组织网站文件的常见方式。第一个图是文件组织结构的鸟瞰图，接下来是将所有文件夹展开后的视图

```
…
<head>
    <meta charset="UTF-8" />
    <title>El Palau de la Música</title>
    <link rel="stylesheet"
    → href="css/style.css" />
</head>
…
```

图 8.2.6　由于 style.css 位于名为 css 的子文件夹中，因此 example.html 中指向样式表的路径应做相应的修改

提示 外部样式表里的 URL 是相对于服务器上该样式表文件的位置的，而不是相对于 HTML 页面的位置的。你将在 10.10 节学习 CSS 背景图像时看到有关的实例。

提示 可以在页面中使用多个 link 元素，从而加载多个样式表文件。如果在不同的文件中有显示声明产生冲突，则后面文件中的规则的优先级更高。

提示 外部样式表中的规则可能被 HTML 文档内的样式覆盖。8.5 节中总结了以不同方式应用样式的相对影响力。

提示 可以通过设置 media 属性将样式表限制在特定类型的输出上，详细信息参见 8.6 节。

提示 HTML 早期版本要求在 link 元素定义中包含 type="text/css"，但 HTML5 不要求这样做，因此可以像代码示例那样忽略它。

8.3　创建嵌入样式表

嵌入样式表是在页面中应用 CSS 的第二种方式。我们在 HTML 页面的 head 部分创建一个 style 元素，其中包含了我们的样式表，如图 8.3.1 所示。由于这些样式只在一个网页里存在，因此不会像外部样式表中的规则那样应用到其他的页面，同时，缓存的好处也不存在了。如上文所述，对于大多数情况，我们推荐使用外部样式表，但理解其他的选择以备不时之需也是很重要的。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>El Palau de la Música</title>
    <style>
    img {
        border: 4px solid red;
    }
    </style>
</head>
<body>
<article>
    <h1>El Palau de la Música</h1>

    <img src="img/palau.jpg" width="250"
    → height="163" alt="El Palau de la
    → Música" />

    <img src="img/tickets.jpg" width="87"
    → height="163" alt="The Ticket Window" />

    <p>I love the <span lang="es">Palau de
    → la Música</span>. It is ornate and
    → gaudy and everything that was
    → wonderful about modernism. It's also
    → the home of the <span lang="es">Orfeó
    → Català </span>, where I learned the
    → benefits of Moscatell.</p>
</article>
</body>
</html>
```

图 8.3.1　使用嵌入样式表时，style 元素及其包围的样式规则通常位于文档的 head 部分。浏览器对页面的呈现方式与使用外部样式表时是一样的，参见图 8.3.2



图 8.3.2　结果与链接到外部样式表完全一样。区别在于，其他网页无法利用这个页面中定义的样式

提 示　当且仅当 style 元素出现在 link 元素后面的时候，嵌入样式表中的样式才会覆盖外部样式表中的样式。详细信息参见 8.5 节。

提 示　嵌入样式表是为页面添加 CSS 的次选方式。（也有例外的情况，如特定条件下拥有极大流量的网站。）

提 示　HTML5 之前的版本要求在 style 开始标签中包含属性 type="text/css"，不过 HTML5 没有要求。因此你可以省略，本书的示例代码中就都没有写。

创建嵌入样式表的步骤

(1) 在 HTML 文档的 head 部分输入 <style>。

(2) 根据需要，定义任意数量的样式规则，有必要的话添加适量 CSS 注释（参见 7.1 节和 7.2 节）。

(3) 输入 </style> 结束嵌入样式表（参见图 8.3.1）。

8.4　应用内联样式

内联样式是在 HTML 中应用 CSS 的第三种方式。不过，应当最后考虑这种方式，因为它将内容（HTML）和表现（CSS）混在了一起，严重地违背了最佳实践，如图 8.4.1 所示。内联样式只影响一个元素，如图 8.4.2 所示，因此使用它将失去外部样式表的关键好处：一次编写，到处可见。设想要对大量 HTML 做简单的文字颜色的改变，就需要对这些页

面逐一进行检查和修改，这也是大家不会经常使用内联样式的原因。

不过，如果你想快速地测试某种样式，以便随后将它从 HTML 中搬到更易于长期维护的外部样式表中（假定你对结果满意），内联样式就能派上用场了。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>El Palau de la Música</title>
</head>
<body>
...
    <img src="img/palau.jpg" width="250"
    → height="163" alt="El Palau de la Música"
    → style="border: 4px solid red" />

    <img src="img/tickets.jpg" width="87"
    → height="163" alt="The Ticket Window" />
...
</body>
</html>
```

图 8.4.1 内联样式规则只影响单个元素（在本例中为第一个 img 元素）



图 8.4.2 只有第一个图像拥有边框。如果要让其他元素也有这种效果，就要在每个 img 元素中单独加上 style="border: 4px solid red"。如你所见，内联样式的效率很低，对全站的内联样式统一进行更新是一件很头疼的事。更好的做法请参见图 8.4.3 和图 8.4.4

```
...
<body>
...
    <img src="img/palau.jpg" width="250"
    → height="163" alt="El Palau de la Música"
    → class="frame" />

    <img src="img/tickets.jpg" width="87"
    → height="163" alt="The Ticket Window" />
...
</body>
</html>
```

图 8.4.3 这是实现图 8.4.2 所示效果的一种较好的方法。为第一个 img 添加一个类名，而不是添加一个内联样式。然后，在外部样式表中为该类创建一条规则（如图 8.4.4 所示）。如果你希望其他的元素拥有这种样式，只需要为其添加 class="frame" 即可。用任何类名都行，frame 只是一个例子

```
.frame {
    border: 4px solid red;
}
```

图 8.4.4 通过使用类选择器，这条规则可以为任何在 HTML 中拥有 class="frame" 的元素（不仅是 img）添加红色的边框。将它包含在外部样式表中，这样就可以很方便地在全站应用这种样式。下一章将介绍选择器的相关知识

应用内联样式的步骤

(1) 在希望进行格式化的 HTML 元素的开始标签中，输入 style="（或者直接在 img 等虚元素的标签内输入，虚元素不需要结束标签）。

(2) 创建一个样式规则，但不要包含花括号和选择器。不需要选择器是因为直接将样式放入目标元素中了。如果你的样式规则包含多条声明，使用分号将不同的声明隔开。

(3) 输入后引号 "。

提 示 注意，不要混淆等号与冒号。由于它们都表示赋值，因而很容易不小心用错。

提示 不要忘记用直引号包围样式定义，以及用分号分隔多条属性定义。

提示 内联样式的优先级高于其他所有样式，除非其他地方与之冲突的样式标记了 !important（参见 8.5 节）。

提示 如果要在内联样式声明中指定字体（参见第 10 章），对于有多个单词的字体名称，要用单引号包围，以避免与 style 元素的双引号发生冲突。不能在这两个地方使用相同类型的引号。

提示 或许内联样式最为常见的使用场景是在 JavaScript 函数中为元素应用内联样式，从而为页面某个部分添加动态行为。当你关注页面来源的时候可能会注意到这些生成的内联样式，如 Firebug 或 Chrome 的开发者工具。在大多数情况下，应用这些样式的 JavaScript 同 HTML 是分离的，因而仍然保持了内容（HTML）、表现（CSS）和行为（JavaScript）分离的原则。

8.5 样式的层叠和顺序

将多个样式规则应用于同一元素的情况并不少见，大型网站更是如此，他们需要更多精力维护 CSS。有时候这些规则针对的是同一条属性。正如 7.4 节中提到的，在规则的特殊性相同的情况下，样式的顺序就成为关键。基本规则是：在其他条件相同的情况下，越晚出现的样式优先级越高（参见图 8.5.1 至图 8.5.5）。以下几条解释了顺序如何决定样式规则的优先级。

- 嵌入样式表（位于 style 元素内）与任何链接的外部样式表之间的关系取决于它们在 HTML 中的相对位置。在两者发生冲突时，如果 link 元素在 HTML 代码中出现得早（图 8.5.2），style 元素就会覆盖链接的样式表（图 8.5.3）；如果 link 元素出现得晚（图 8.5.4），其中的样式及其包含的任何导入样式表就会覆盖 style 元素的规则（图 8.5.5）。
- 内联样式（实际上直接应用于元素）在外部样式表和嵌入样式表之后。由于顺序最靠后，其优先级是最高的。一旦应用到任何地方，都会覆盖与之冲突的其他样式。
- 关于相互冲突的样式的顺序对优先级的影响，有一种例外情况，就是标记 !important 的样式总是具有最高的优先级，无论它出现在最前、最后，还是中间。例如，`p { margin-top: 1em !important; }`。不过，要尽量避免这种用法，极端情况除外。几乎在全部情况下，使用选择器都能达到同样的效果。此外，!important 让声明变得太强，如果要覆盖这样的样式，就不得不借助于更长的规则。

```
img {
    border: 4px solid red;
}


/*
上面是对以下编写方法的简化：


img {
    border-width: 4px;
    border-style: solid;
    border-color: red;
}
*/
```

图 8.5.1 这是我们熟悉的外部样式表的例子 style.css，是从前面搬过来的

```
...
<head>
    <title>El Palau de la Música</title>

    <link rel="stylesheet" href="style.
css" />

    <style>
    img {
       border-style: dashed;
    }
    </style>
</head>
...
```

图 8.5.2　在这个例子中，嵌入样式表出现在最后，因此，它的样式的优先级高于 style.css 里的样式（如果冲突的样式规则具有相同的继承性和特殊性的话）。嵌入样式表会覆盖 style.css 中的边框样式，将实线改为虚线，但不会影响宽度和颜色（参见图 8.5.3）



图 8.5.3　style 元素中定义的虚线边框优先于链接的 style.css 中定义的实线边框，不过保留了 style.css 中指定的边框宽度和颜色

明白了吗？不过，由于你只会使用一个外部样式表，且会避免使用 !important（是吧），因此好的一面是，你不必担心我在上面讲解的这些规则。不过，如果你将来看到其他人的代码并不是像你这样编写的，了解上述知识就很有用了。

对你来说，顺序规则影响最大的情形是：如果网页包括一个以上的 link 元素，不同的外部样式表中的规则如有冲突，后出现的样式表中的规则优先。

```
...
<head>
    <title>El Palau de la Música</title>
    <style>
    img {
       border-style: dashed;
    }
    </style>
    <link rel="stylesheet"
    →href="style.css" />
</head>
...
```

图 8.5.4　这里，链接的样式表最后出现，其优先级高于 style 元素中的规则（其他情况都相同）



图 8.5.5　style.css 样式表中定义的实线边框优先于内部 style 元素中定义的虚线边框

> **提示**　建议不要使用 @import，这主要是出于性能上的考虑。如果用了 @import，当引入的样式后面有与之冲突的样式，则后出现的样式会覆盖这些引入的样式，与预期一致。

> **提示**　有一种使用 !important 的合理情形。有时网页会包含一些你无法修改的 HTML，例如来自第三方服务的新闻源。如果这些 HTML 中有一些内联样式与你的设计不符，你就可以在自己的样式表中使用 !important 覆盖这些样式。

8.6 使用与媒体相关的样式表

可以指定一个只用于特定输出的样式表，如只用于打印，或只用于在浏览器中查看屏幕。例如，可以创建一个具有打印和屏幕版本共有特征的通用样式表，再创建单独的打印样式表和屏幕样式表，分别包含只用于打印的属性和只用于屏幕显示的属性。

指定与媒体相关的样式表的步骤

(1) 在 link 或 style 元素的开始标签中添加 media="*output*"，其中 *output* 可以是 print、screen 或 all（尽管还有其他一些选项，但这些是最常见的），参见图 8.6.1。多个值之间用逗号分隔。

(2) 也可以在样式表中使用 @media 规则，参见图 8.6.2。这种方法不需要在 link 元素中指定媒体类型。

提示　有 9 种可能的输出类型：all、aural、braille、handheld、print、projection、screen、tty 和 tv。浏览器对它们的支持程度各不相同（大多只有少量的支持）。实际上，用得上的只有 screen 和 print（或许还应加上 all），浏览器对它们的支持情况都很好。另一方面（可以这么说），设备对 handheld 的支持程度并不高，因此在为移动设备设计时，通常使用 screen 而不是 handheld（参见第 12 章）。projection 类型是为投影仪和其他类似的设备准备的，Opera 的投影模式 Opera Show 支持这种类型。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>El Palau de la Música</title>
    <link rel="stylesheet" href="style.css"
    → media="screen" />
    <link rel="stylesheet" href="print.css"
    → media="print" />
</head>
<body>
<article>
    <h1>El Palau de la Música</h1>

    <img src="img/palau.jpg" width="250"
    → height="163" alt="El Palau de la
    → Música" />

    <img src="img/tickets.jpg" width="87"
    → height="163" alt="The Ticket Window" />

    <p>I love the <span lang="es">Palau de
    → la Música</span>. It is ornate and
    → gaudy and everything that was
    → wonderful about modernism. It's also
    → the home of the <span lang="es">Orfeó
    → Català</span>, where I learned the
    → benefits of Moscatell.</p>
</article>
</body>
</html>
```

图 8.6.1　通过对 link 元素添加 media 属性，可以将样式表限于特定输出。在这个例子中，使用浏览器查看页面时，style.css 会起作用（由于使用了 media="screen"）；打印页面时，print.css 会起作用（由于使用了 media="print"）。如果第一个 link 元素包含了 media="all" 的设置，或者没有 media 属性，style.css 中的规则也会应用到打印页面

8

```
/* 针对所有媒体的样式 */
img {
    border: 4px solid red;
}

p {
    color: orange;
    font-style: italic;
}

/* 打印样式表 */
@media print {

    body {
       /* 加大字号 */
       font-size: 25pt;
    }

    p {
       /* 黑色的十六进制值 */
       color: #000;
    }

    img {
       /* 不显示图像 */
       display: none;
    }

}
```

图 8.6.2　样式表中的 @media 规则是指定其他媒体类型的另一种方式（参见第 12 章）。在这个例子中，上面是影响所有媒体类型（包括打印）的样式，下面是专门用于打印的样式，样式规则放在 @media print { } 中。在浏览器中查看这个页面，效果跟本章其他图一样，只是文字是橙色和倾斜的。这个页面的打印版如图 8.6.3 所示

提 示　联合使用 CSS3 引入的媒体查询和这里介绍的媒体输出类型可以形成强大的威力。通过这些技术，可以根据输出设备的属性确定需要应用到页面的样式。例如，可以让页面在较窄的屏幕（如智能手机屏幕）上呈现为一种样式，在较宽的屏幕（如笔记本屏幕）上呈现为另一种样式。现在，设备的种类越来越多，媒体查询已经变成代码编写人员的一种重要工具。第 12 章将向你介绍如何构建可以在各种设备上正常运行的网站。关于媒体查询的内容是从 12.4 节开始介绍的。



图 8.6.3　Chrome 浏览器的打印预览功能可以显示图 8.6.1 中页面打印出来的样子。文本变大了，不会显示图像，段落文本是加粗的斜体，而不是橙色的。font-style: italic 这条声明也应用到了打印模式，因为打印样式表没有指定其他的 font-style

8.7　借鉴他人的灵感

我们在第 2 章中学习了如何查看网页的源代码。查看他人的 CSS 也不难。

查看其他设计人员的 CSS 代码

(1) 首先查看页面的 HTML 代码（如图 8.7.1 所示）。关于查看 HTML 源代码的更多信息，参见 2.8 节。

如果 CSS 代码是嵌入样式表，就已经能看到了。

(2) 如果 CSS 是外部样式表，就在 HTML 中找到对它的引用，再点击文件名（参见图 8.7.1）。样式表就显示在浏览器窗口中了（如图 8.7.2 所示）。如果愿意，可以复制这些代码，再粘贴到文本编辑器中。



图 8.7.1　查看包含你感兴趣的样式表的 HTML 页面源代码，再点击样式表的文件名

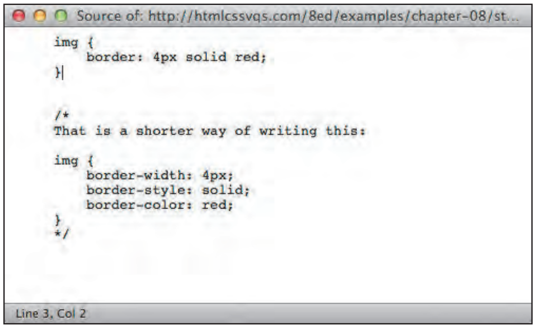


图 8.7.2　样式表显示在浏览器窗口中，img 规则下面包含了 CSS 注释（因此确保注释整洁）

提 示　同 HTML 一样，可以从其他设计人员的代码中借鉴灵感，然后编写自己的样式表。不过，查看其他人的代码时也要留心。这些代码放在万维网上并不意味着它们就是编写某一效果的代码的最佳方式（尽管其作者希望如此）。

提 示　现代浏览器允许像图中显示的那样直接在 HTML 代码中点击样式表的名称。要在旧的浏览器中查看样式表，可能需要复制 link 元素中显示的 URL，再粘贴到浏览器的地址栏（替换 HTML 文件名），然后按回车键。如果样式表的 URL 是相对地址（参见 1.7 节），就需要将网页的 URL 和样式表的相对 URL 结合在一起，形成样式表的完整 URL。

提 示　使用现代浏览器提供的开发者工具，可以快速查看页面的 CSS。大多数浏览器都捆绑了这类工具。Firefox 中有一个实现此功能的扩展，名为 Firebug（参见第 20 章）。

8

第 9 章

定义选择器

本章内容

- 构造选择器
- 按名称选择元素
- 按类或 ID 选择元素
- 按上下文选择元素
- 选择第一个或者最后一个子元素
- 选择元素的第一个字母或者第一行
- 按状态选择链接元素
- 按属性选择元素
- 指定元素组
- 组合使用选择器

正如 7.1 节中看到的，CSS 样式规则有两个主要部分。**选择器**决定将格式化应用到哪些元素，而**声明**则定义要应用的格式化。在本章中，我们将学习如何定义 CSS 选择器。

最简单的选择器可以对给定类型的所有元素（如所有的 h2 标题）进行样式化，有的选择器允许我们根据元素的类、上下文、状态等来应用格式化规则。随着你对本章内容的学习，你会了解更多类型的选择器。请记住，我们可以在一个选择器中组合使用多个选择器，例如，编写类选择器和属性选择器的复合选择器。

定义选择器以后，可以根据第 10 章 ~ 第 16 章的讲解创建声明（包括实际的属性和值）。在此之前，我们将在例子中使用非常简单且意义明确的 `{color: red;}`。

9.1 构造选择器

选择器决定样式规则应用于哪些元素。例如，如果要对所有的 p 元素添加 Georgia 字体、12 像素高的格式，就需要创建一个只识别 p 元素而不影响代码中其他元素的选择器。如果要对每个区域中的第一个 p 设置特殊的缩进格式，就需要创建一个稍微复杂一些的选择器，它只识别作为页面中每个区域的第一个元素的 p 元素。

选择器可以定义五个不同的标准来选择要进行格式化的元素。

- 元素的类型或名称，如图 9.1.1 所示。
- 元素所在的上下文，如图 9.1.2 所示。
- 元素的类或 ID，如图 9.1.3 和图 9.1.4 所示。
- 元素的伪类或伪元素，如图 9.1.5 所示（放心，我将在后面解释伪类和伪元素）。
- 元素是否有某些属性和值，如图 9.1.6 所示。

为了指出目标元素，选择器可以使用这五个标准的任意组合。在大多数情况下，只使用一个或两个标准即可。另外，如果要对几组不同的元素应用相同的样式规则，可以将相同的声明同时应用于几个选择器（参见 9.9 节）。

本章其余部分将详细解释如何定义选择器。

```
要选择的元素的名称
h1 {
   color: red;
}
```

图 9.1.1　最简单的选择器类型就是要格式化的元素的类型的名称（在这个例子中为 h1）

```
上下文
   要选择的元素的名称
h1 em {
   color: red;
}
```

图 9.1.2　这个选择器使用上下文。这个样式只应用于 h1 元素中的 em 元素。其他地方的 em 元素不受影响

```
  类
.error {
   color: red;
}

  ID
#gaudi {
   color: red;
}
```

图 9.1.3　第一个选择器选择所有属于 error 类的元素。换句话说，就是所有在开始标签中包含 class="error" 的元素。第二个选择器选择 id 为 gaudi 的那个元素，也就是在开始标签中指定 id="gaudi" 的元素。回忆一下，一个 id 在每个页面中只能出现一次，而一个 class 可以出现任意次数。这是我们推荐 class 选择器而不推荐 ID 选择器的主要原因，你可以针对尽可能多的元素重用 class 选择器的样式

```
要选择的元素的名称
               类
strong.error {
   color: red;
}
```

图 9.1.4　通过在 class 或 id 选择器前面添加目标元素的名称，可以增强选择器的特殊性。在这个例子中，第一个选择器仅选择属于 error 类的 strong 元素，而不是属于 error 类的任何元素。除非确有必要，一般不要使用这种方法。图 9.1.3 中的选择器是更好的方法，因为它比较灵活

```
名称
   伪类
a:link {
   color: red;
}
```

图 9.1.5　在这个例子中，选择器选择属于 link 伪类的 a 元素（即页面上还未访问的链接）

```
名称
   属性
a[title] {
   color: red;
}

名称
   属性              值
a[href="http://www.wikipedia.org"] {
   color: red;
}
```

图 9.1.6　可以在选择器中使用方括号添加关于目标元素的属性或值的信息，第一个例子针对的是所有具有 title 属性的 a 元素，第二个例子针对的只是指向维基百科页面的 a 元素

提示　我们建议你尽量不要使用 ID 选择器，我会在 9.3 节详细解释原因。

提示　我们会在 9.8 节了解到，选择器不仅限于针对一个属性的确切值，图 9.1.6 就是这种情况。

提示　编写 CSS 的一个重要目标就是让选择器尽可能的简单，仅保持必要的特殊性。应该充分利用样式的层叠特性，即元素的后代会继承祖先元素的样式。同时，应该将页面中的通用设计元素挑选出来，编写一个选择器（如类名），从而可以在全站不同的元素上共享该样式。这样，样式表通常会变得更小，也更容易维护。随着你构建页面经验的积累，你会越来越深地体会到这些技巧，不过现在先提出来，有助于日后的理解。

9.2　按名称选择元素

选择要格式化的元素（参见图 9.2.1），最常用的标准可能是元素的名称（也就是大家平常说的**类型选择器**，因为你指定了应用样式的元素的类型）。例如，可能要让所有的 h1 元素以大字号、粗体显示，让所有的 p 元素以无衬线字体显示。

按照类型选择要格式化的元素

(1) 输入 *selector*，其中 *selector* 是目标元素的类型名称（不含任何属性），如图 9.2.2 所示。

(2) 输入 {。

(3) 输入你希望应用到选中元素的样式，这要用属性 : 值对的形式表示。我们从下一章才开始讲解 CSS 属性和值。

(4) 输入 } 结束样式规则。

```
<!DOCTYPE html>
<html lang="en">
<head>
...
</head>
<body>
...
<article class="architect">
    <h1>Antoni Gaudí</h1>
    <p>Many tourists are drawn to Barcelona to see Antoni Gaudí's incredible architecture.</p>

    <p>Barcelona <a href="http://www.gaudi2002.bcn.es/english/" rel="external">celebrated the 150th
    → anniversary</a> of Gaudí's birth in 2002.</p>

    <h2 lang="es">La Casa Milà</h2>
    <p>Gaudí's work was essentially useful. <span lang="es">La Casa Milà</span> is an apartment
    → building and <em>real people</em> live there.</p>

    <h2 lang="es">La Sagrada Família</h2>
    <p>The complicatedly named and curiously unfinished Expiatory Temple of the Sacred Family is
    → the <em>most visited</em> building in Barcelona.</p>
</article>
...
```

图 9.2.1　这段 HTML 代码包含两个 h2 元素（或许你对 lang 属性感到疑惑。它指明内容使用的是页面默认语言以外的另一种语言。页面默认语言是在每页开头处紧跟在 DOCTYPE 后面的 html 元素中指定的。在这个例子中，每个 h2 上的 lang="es" 表示其内容为西班牙语）

注意，在本章后面的章节中，再谈到构造选择器的步骤时我不会重述第 (2) ~ (4) 步的内容了，不过对于创建完整的样式规则，这些步骤都是必需的。更多信息请参见 7.1 节。

```
h2 {
    color: red;
}
```

图 9.2.2　这个选择器会选择文档中所有的 h2 元素，并让它们显示为红色，参见图 9.2.3

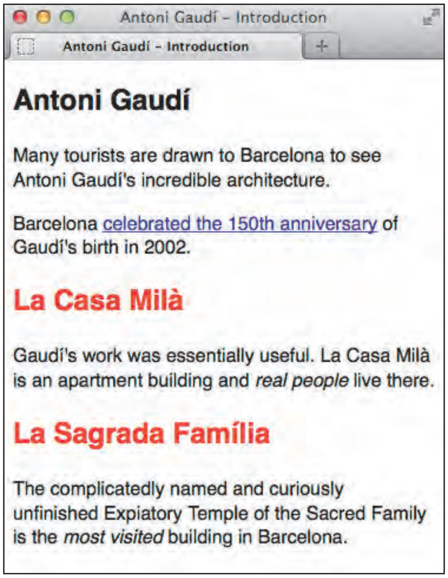


图 9.2.3　所有的 h2 元素都显示为红色

> **提示**　除非指定其他情况（使用本章其余部分介绍的技术），指定类型的所有元素都将被格式化，无论它们出现在文档的什么位置。

> **提示**　并非所有的选择器都需要指定元素的名称。如果对某一类的元素进行格式化，而不管属于这个类的元素的类型，就可以从选择器中去掉元素名称。下一节将解释具体怎么做。

> **提示**　通配符 *（星号）匹配代码中的任何元素名称。例如，使用 * { border: 2px solid green; } 会让每个元素都有一个 2 像素宽的绿色实线边框！因为匹配范围太广，会让浏览器加载页面变慢，因此应该谨慎使用通配符。实际适合使用通配符的情况比较少。

> **提示**　可以在一个选择器中使用一组元素名称，名称之间用逗号分隔。更多细节参见 9.9 节。

9.3　按类或 ID 选择元素

如果已经在元素中标识了 class（参见图 9.3.1）或 id（参见最后一条提示），就可以在选择器中使用这个标准，从而只对已标识的元素进行格式化（参见图 9.3.2）。不过推荐使用类选择器，一会儿我会解释理由。

1. 按类选择要格式化的元素

(1) 输入 .（点号）。

(2) 不加空格，直接输入 *classname*，这里的 *classname* 标识希望应用样式的类。

2. 按 id 选择要格式化的元素

(1) 输入 #（井号）。

(2) 不加空格，直接输入 *id*，这里的 *id* 唯一标识希望应用样式的元素。

> **提示**　可以单独使用 class 和 id，也可以同其他选择器标准混在一起使用。例如，.news { color: red; } 会影响所有属于 news 类的元素，而 h1.news { color: red; } 只会影响属于 news 类的 h1 元素。除非必须特别针对目标元素，最好不要在 id 或 class 选择器中添加元素名称。

9

```
...
<article id="gaudi" class="architect">
    <h1>Antoni Gaudí</h1>
    <p>Many tourists are drawn to Barcelona to see Antoni Gaudí's incredible architecture.</p>

    <p>Barcelona <a href="http://www.gaudi2002.bcn.es/english/" rel="external">celebrated the 150th
    ➝ anniversary</a> of Gaudí's birth in 2002.</p>

    <h2 lang="es">La Casa Milà</h2>
    <p>Gaudí's work was essentially useful. <span lang="es">La Casa Milà</span> is an apartment
    ➝ building and <em>real people</em> live there.</p>
    ...
</article>

<p>This paragraph doesn't have <code>class="architect"</code>, so it isn't red when the CSS is
➝ applied.</p>

<article class="architect">
    <h1>Lluís Domènech i Montaner</h1>
    <p>Lluís Domènech i Montaner was a contemporary of Gaudí.</p>
    ...
</article>
...
```

图 9.3.1　有两个 class 为 architect 的 article 元素。它们之间还有一个不含 class 的短段落

```
.architect {
    color: red;
}
```

图 9.3.2　这个选择器会选择 class 为 architect 的元素。在这个例子中，它们都是 article 元素，不过，可以将类应用于任何元素。如果只想对有这个类的 article 元素应用这种样式，可以将选择器写为 article.architect。不过，这样做通常会使特殊性高于你所需要的程度

提 示　注意，在图 9.3.1 和图 9.3.2 中使用了表达内容含义的 class 名称（architect），而不是命名为 red。尽管也有极端情况，但最好避免创建描述事物如何显示的 class 名称（一般称为**描述性类名**），因为你可能在将来改变样式（例如，在本节示例中让文本显示为绿色）。

提 示　如果要定位的元素有多个类名，可以在选择器中将它们连在一起，就像 .architect.bio { color: blue; } 这样。任何 .architect 或 .bio 选择器的规则仍会应用于该元素，但 .architect.bio 的规则的特殊性更高，因此如果有样式冲突，.architect.bio 的规则的优先级更高。注意类名之间没有空格。如果有空格，就会针对任何 architect 类的元素嵌套的 bio 类元素设置样式（参见 9.4 节）。

提 示　如果图 9.3.2 中写的是 #gaudi { color: red; }，就只有第一个 article 的文本会显示为红色，因为它是唯一包含 id="gaudi" 的元素。每个 id 都必须是唯一的，因此，不能在关于 Lluís Domènech i Montaner 的 article 上再用这个 id。

提示 关于在 HTML 代码中为元素指定类的信息，参见 3.14 节。



图 9.3.3 属于 architect 类的 article 显示为红色，而两个 article 元素之间的 p 元素没有显示为红色（或许你对链接显示为蓝色感到疑惑。链接显示为蓝色是浏览器的默认样式，可以编写自己的样式覆盖它）

类选择器与 ID 选择器的比较

要在 class 选择器和 id 选择器之间作出选择的时候，建议尽可能地使用 class 选择器。这主要是因为我们可以复用 class 选择器。有人提议完全不使用 id 选择器，我赞同这种观点，并且在自己的工作中始终遵循这一点，但最终的决定权掌握在网站开发人员手中。id 选择器会引入下面两个问题。

- 与它们关联的样式不能在其他元素上复用（记住，在一个页面中，一个 id 只能出现在一个元素上）。这会导致在其他元素上重复样式，而不是通过 class 共享样式。
- 它们的特殊性比 class 选择器要强得多。这意味着如果要覆盖使用 id 选择器定义的样式，就要编写特殊性更强的 CSS 规则。如果数量不多，可能还不难管理。如果处理规模较大的网站，其 CSS 就会变得比实际所需的更长、更复杂。

随着你处理的 CSS 越来越多，你对这两点的理解也会更加清晰。（另一方面，有人喜欢使用 id 选择器的原因之一就是使用它们一眼就能看出元素是唯一的。不过，依照个人经验，这点好处还不足一均衡它带来的弊端。另外，万一你哪天改变了网站设计，原先唯一的元素不再唯一，id 选择器就不合适了。）

因此，推荐的做法是寻找能将共享样式结合进一个或多个 class 的机会，从而可以对它们进行复用，同时，如果确实要使用 id 选择器，也应该尽量少用。这样，你的样式表会比较短，更易于管理。

需要说明的是，id 选择器在 HTML 中仍具有重要的作用。通过它们可以在页面中定义锚（参见 6.2 节），在编写 JavaScript 为特定的页面元素应用特殊行为时它们尤其具有重要的价值（JavaScript 是一个单独的主题，本书不会作过多介绍）。

9.4　按上下文选择元素

在 CSS 中，可以根据元素的祖先、父元素或同胞元素来定位它们（参见 1.3 节中的“父元素和子元素”），如图 9.4.1 ~ 图 9.4.4 所示。

```
...
<article class="architect">
    <h1>Antoni Gaudí</h1>
    <p>Many tourists ... </p>
    <p>Barcelona ... </p>

    <section>
       <h2 lang="es">La Casa Milà</h2>
       <p>Gaudí's work ... </p>
    </section>

    <section>
       <h2 lang="es">La Sagrada Família</h2>
       ...
    </section>
</article>
...
```

图 9.4.1　这里对 article 的一部分使用了 section 元素，从而可以演示几代元素的关系。同时，对文字内容进行了压缩，这样可以更容易看清元素之间的关系。注意，在这个代码片段中，有两个第二代 p 元素，它们由类名为 architect 的 article 直接包含，另外还有一个第三代 p 元素，位于第一个 section（包含在 article 里）。在完整的代码中还有一个第三代 p 元素，不过在这里没有显示出来。这里，所有 h2 的实例也是第三代元素

```
.architect p {
    color: red;
}
```

图 9.4.2　这里组合使用了类选择器和类型选择器。.architect 和 p 之间的空格表示这个选择器会寻找任何作为 architect 类元素后代（无论是第几代）的 p 元素。实际效果参见图 9.4.4，此外，图 9.4.3 提供了几种实现同样效果的其他方式

```
/* 得到相同效果的其他方式
------------------------------------- */
/* 是任意article祖先的所有p元素，
   这是三个中特殊性最低的一个 */
article p {
    color: red;
}

/* 属于architect类article元素的祖先的任意p元
   素，是三个中特殊性最高的一个 */
article.architect p {
    color: red;
}
```

图 9.4.3　第一个例子中的选择器（article p { }）比图 9.4.2 中的选择器和它后面的选择器（article.architect p { }）的特殊性都要低。这里演示的第二个例子比其他两个例子的特殊性都要高，不过在实践中，在类名（尤其是 ID 名）之前添加元素名通常会让特殊性比实际需要的要高

祖先（ancestor）是包含目标元素（**后代**，descendant）的任何元素，不管它们之间隔了多少代。（父元素是直接包含另一个元素的元素，它们之间只隔一代，被包含的元素称为子元素。）通常对元素的子元素增加缩进，从而可以清晰地看到它们之间的关系（如图 9.4.2 所示）。缩进对页面的显示没有任何影响。

要实现某种效果，通常有多种构建选择器的方式可供选择。具体取决于你需要多大的特殊性（如图 9.4.3 所示）。

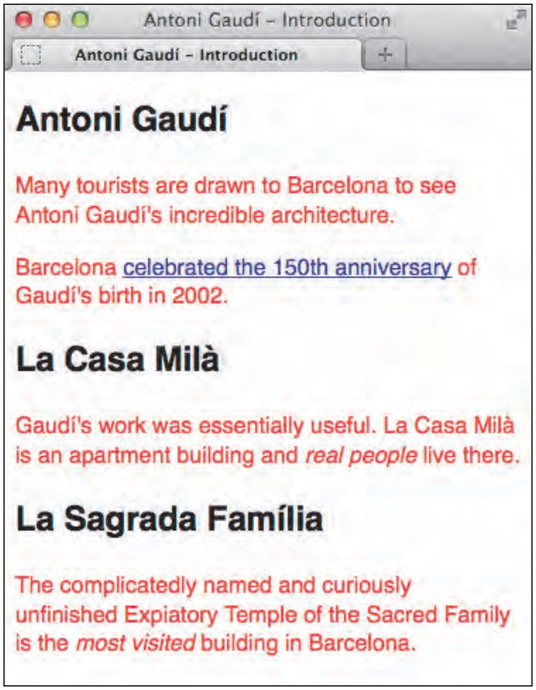


图 9.4.4　所有包含在属于 architect 类的元素里的 p 元素都是红色的，即使它们同时也包含在其他属于 architect 类的元素。图 9.4.2 和图 9.4.3 中每条样式规则的结果都显示在这里

1. 按祖先元素选择要格式化的元素

(1) 输入 *ancestor*，这里的 *ancestor* 是希望格式化的元素的祖先元素的选择器。

(2) 输入一个空格（必不可少）。

(3) 如果需要，对后续的每个祖先元素重复第 (1) 步和第 (2) 步。

(4) 输入 *descendant*，这里的 *descendant* 是希望格式化的元素的选择器。

提示　我们通常将基于元素祖先的选择器称为后代选择器，不过 CSS3 将其重新命名为**后代结合符**。（有些人仍然使用“选择器”这一名称。）

提示　不要对图 9.4.3 中的 article.architect 部分感到疑惑。记住这只表示“class 等于 architect 的 article”。因此 article.architect p 表示“包含在 class 等于 architect 的 article 元素里的任何 p 元素”。相比之下，特殊性低一些的 .architect p 表示“包含在 class 等于 architect 的**任意**元素里的所有 p 元素”，参见图 9.4.3。

2. 按父元素选择要格式化的元素

上面的例子展示了后代结合符。CSS 也有**子结合符**，从而可以为父元素的**直接后代**（即子元素）定义样式规则。在 CSS3 之前，它们称为**子选择器**。

(1) 输入 *parent*，这里的 *parent* 是直接包含待格式化元素的元素的选择器。

(2) 输入 >（大于号），参见图 9.4.5。

(3) 如果需要，对后续每代父元素重复第 (1) 步和第 (2) 步。

(4) 输入 *child*，这里的 *child* 是要格式化的元素的选择器。

```
.architect > p {
    color: red;
}
```

图 9.4.5　这个选择器仅选择 architect 类元素的子元素（而非子子元素、子子子元素等）的 p 元素。包含于任何其他元素的 p 元素均不会被选中，实现参见图 9.4.6

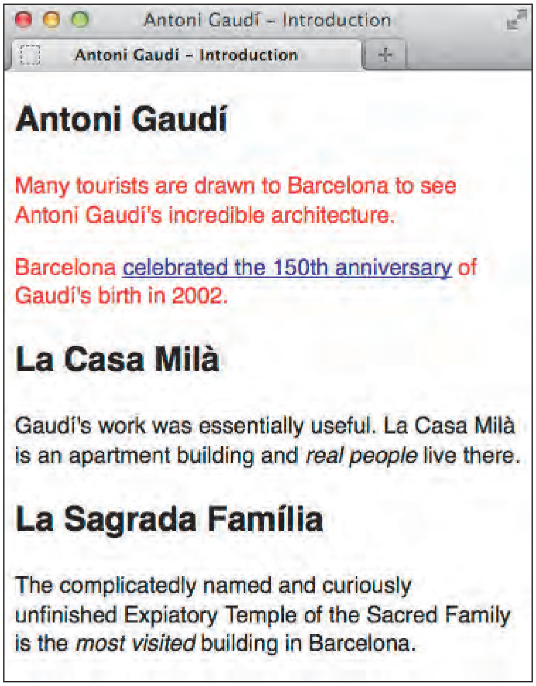


图 9.4.6　只有头两个 p 元素是 architect 类元素的子元素。另两个 p 元素是 article 里 section 元素的子元素。这个例子使用的 HTML 代码见图 9.4.1

提示　正如你在后代结合符中看到的，可以忽略 class 前面的元素名称。例如，在上面的例子中使用 article.architect > p { color: red; } 会产生同样的效果，但这个选择器的特殊性更高。要想比以上两种方法的特殊性更低（图 9.4.5 就是个例子），应该忽略整个类别，就像图 9.4.5 中的 article > p{ color: red; }。本章剩余部分的一些例子也可以按类似的方式进行简化。你已经了解了如何进行简化，因此不必再一一列举这些替代方案，但应记住，通常最好保持**较低**的特殊性，让样式更易于复用。

提示　也可以在子结合符中使用 id 选择器，但推荐尽量使用特殊性更低的选择器（如类选择器）。

提示　学习本章内容同时可参考 1.3 节中的“父元素和子元素”。

3. 按相邻同胞元素选择要格式化的元素

下面继续讲我们的“家族”主题。**同胞**（sibling）元素是拥有同一父元素的任何类型的子元素。**相邻同胞元素**（adjacent sibling）是直接相互毗邻的元素，即它们之间没有其他的同胞元素。在下面这个简略的例子中，h1 和 p 是相邻同胞元素，p 和 h2 是相邻同胞元素，而 h1 和 h2 则不是相邻同胞元素。不过，它们都是同胞元素（也是 body 元素的子元素）。

```
...
<body>
    <h1>...</h1>
    <p>...</p>
    <h2>...</h2>
</body>
</html>
```

CSS 相邻同胞结合符（adjacent sibling combinator）可以选择直接跟在指定的同胞元素后面的同胞元素，如图 9.4.7 和图 9.4.8 所示。关于 CSS3 中新出现的**普通同胞结合符**（general sibling combinator），参见最后一条提示。

(1) 输入 *sibling*，这里的 *sibling* 是包含在同一父元素中的、直接出现在目标元素前面的元素的选择器。（它们不必是同一种元素类型，只要它们彼此直接相邻就行。）

(2) 输入 +（加号）。

(3) 如有需要，对每个后续的同胞元素重复第 (1) 步和第 (2) 步。

(4) 输入 *element*，这里的 *element* 是要格式化的元素的选择器。

```
.architect p+p {
    color: red;
}
```

图 9.4.7 相邻同胞结合符只选择直接跟在同胞 p 元素之后的元素



图 9.4.8 只有直接跟在同胞 p 元素后面的 p 元素显示为红色。如果后面还有第三个、第四个以及更多的段落，它们也将显示为红色。例如，如果要对除第一个段落以外的所有段落进行缩进，相邻同胞结合符就很有用

提 示 我们也可能会用到**普通同胞结合符**，通过它可以选择那些并非直接出现在另一同胞元素后面的同胞元素。它与相邻同胞结合符的唯一区别是使用 ~（波浪号）代替 + 分隔同胞元素。例如，h1 ~ h2 { color: red; } 会让任何属于同一父元素的同胞 h1 后面的 h2 元素显示为红色（它们可以直接相邻，也可以不直接相邻）。

9.5 选择第一个或最后一个子元素

上一节解释了如何选择作为另一个元素的子元素的元素，举的例子是 .architect > p，选择所有作为 architect 类元素的子元素的段落。不过，有时需要选择仅作为某元素第一个或最后一个子元素的元素。这时，就要用到 :first-child 和 :last-child 伪类（前者参见图 9.5.1 ~ 图 9.5.3，后者参见图 9.5.1、图 9.5.4 和图 9.5.5）。

```
…
<p>A partial list of Gaudí's projects
→ follows:</p>

<ul>
    <li lang="es">La Casa Milà</li>
    <li lang="es">La Sagrada Família</li>
    <li>College of the Teresians <span lang=
    → "es">(Colegio Teresiano)</span></li>
    <li>Park Güell</li>
</ul>
…
```

图 9.5.1 我们将在第 15 章讲解列表，不过在这里你只需要知道，这是一个无序列表（ul），每个列表项目（li）都是其子元素。这是使用 :first-child 和 :last-child 规定样式的常见情形，不过通常是用于添加一个边框，而不是简单地改变一下文字颜色

```
li:first-child {
    color: red;
}
```

图 9.5.2 这个选择器会选择作为父元素的第一个子元素的 li 元素

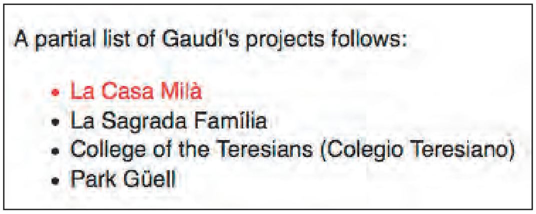


图 9.5.3　因为第一个 li 元素是 ul 元素的第一个子元素，所以显示为红色。缩进和项目符号是浏览器针对无序列表的默认样式

```
li:last-child {
    color: red;
}
```

图 9.5.4　这个选择器只选择作为父元素的最后一个子元素的 li 元素

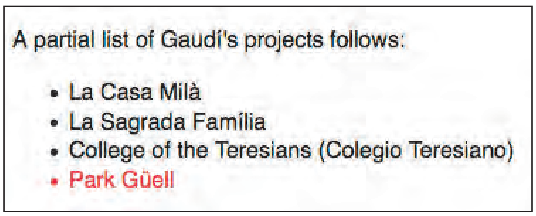


图 9.5.5　这种情况下只有最后一个 li 元素显示为红色

有的人在一开始没有真正弄清楚这些伪类的工作原理。他们以为 li:first-child 这样的选择器会选择 li 元素的第一个子元素，而 li:last-child 会选择 li 元素的最后一个子元素。如果是这样的话，“(Colegio Teresiano)”在两个例子中就都是红色的（参见图 9.5.3 和图 9.5.5），因为它们所在的 span 既是某个 li 的第一个元素，也是其最后一个元素（参见图 9.5.1）。实际情况是，这些伪类选择的是作为第一个子元素或最后一个子元素的元素（在这个例子中就是 li）。

选择某元素的第一个或最后一个子元素进行格式化

(1) 这一步可选，输入代表我们想应用样式的第一个或者最后一个子元素（如 p 或 .news），参见最后一条提示。接下来**千万不要有空格**。

(2) 如果选择的是第一个子元素就输入 :first-child（参见图 9.5.2），如果选择的是最后一个子元素，就输入 :last-child（参见图 9.5.4）。

提示　在下一节的补充材料中我们会更详细地探讨伪类。

提示　:last-child 伪类被添加至 CSS 的时间要晚于 :first-child。因此，IE9 之前的 Internet Explorer 并不支持 :last-child。好在通常你可以绕过这一限制，因为 IE8 及之后的版本都支持 :first-child。这里介绍一种常见的场景（其中提到的外边距在第 11 章才会讲到）。假设你希望每个 li 的下边都有 20 像素的外边距，但最后一个 li 除外。你可以设置 li { margin-bottom: 0; margin-top: 20px; }，让每个列表项目都拥有 20 像素的上外边距，没有下外边距。然后使用 li:first-child { margin-top: 0; } 取消第一个元素的上外边距。这相当于反过来编写代码，但可以达到完全一样的效果。

提示　在伪类之前添加其他选择器可以使其特殊性更高。例如 .architect h1:first-child { color: red; } 仅对作为 architect 类元素的第一个子元素的 h1 应用样式。

提 示　尽管第 (1) 步中指定选择符这一做法是很常见的，但这并非必需。例如，仅仅使用 :first-child { color: red; }，可以为任何作为另一个元素的第一个子元素的元素应用样式。在图 9.5.1 所示的例子中，这样做会让段落（body 的第一个子元素）、第一个 li（ul 的第一个子元素）和“(Colegio Teresiano)”（它所在的 span 是一个 li 的第一个元素）变成红色。

9.6　选择元素的第一个字母或者第一行

我们可以分别使用 :first-letter 和 :first-line 伪元素只选择元素的第一个字母（参见图 9.6.1、图 9.6.2 和图 9.6.3）或第一行（参见图 9.6.1、图 9.6.4 和图 9.6.5）。

```
...
<article class="architect">
    <h1>Antoni Gaudí</h1>
    <p>Many tourists are drawn to Barcelona
    → to see Antoni Gaudí's incredible
    → architecture.</p>
    <p>Barcelona <a href="http://
    → www.gaudi2002.bcn.es/english/"
    → rel="external">celebrated the 150th
    → anniversary</a> of Gaudí's birth in
    → 2002.</p>

    <h2 lang="es">La Casa Milà</h2>
    <p>Gaudí's work was essentially
    → useful...</p>
    ...
</article>
...
```

图 9.6.1　很容易判断 :first-letter 定位的第一个字母，但 :first-line 会影响哪些单词只有在浏览器里查看页面，看到内容的排布情况时才能知道（如图 9.6.5 所示）。HTML 本身无法定义单词位于哪一行

```
p:first-letter {
    color: red;
    font-size: 1.4em; /* make letter
    → larger */
    font-weight: bold;
}
```

图 9.6.2　下面的选择器只选择每个 p 元素的第一个字母

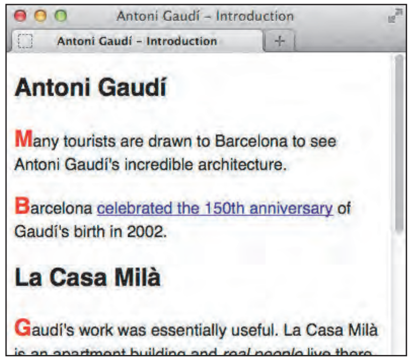


图 9.6.3　first-letter 选择器可用来实现每段首字母大写的效果

```
p:first-line {
    color: red;
}
```

图 9.6.4　下面的选择器只选择每个 p 元素的第一行

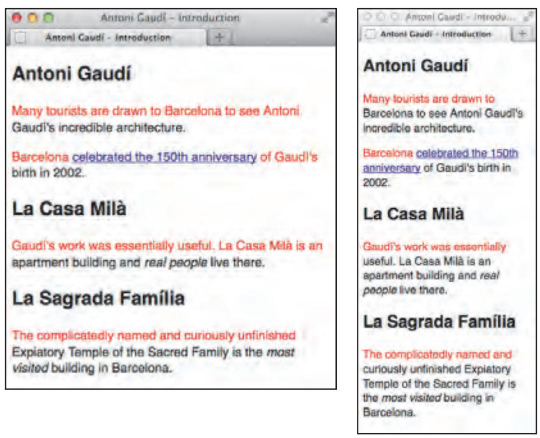


图 9.6.5　调整浏览器窗口会改变每段第一行的具体内容，但选择器选中的始终是第一行

1. 选择元素的第一个字母

(1) 输入 *element*，这里的 *element* 是要对其第一个字母进行格式化的元素的选择器。

(2) 输入 :first-letter 以选择第 (1) 步中引用的元素的第一个字母。

2. 选择元素的第一行

(1) 输入 *element*，这里的 *element* 是要对其第一行进行格式化的元素的选择器。

(2) 输入 :first-line 以选择第 (1) 步中引用的元素的第一行。

提示 可以将 :first-letter 和 :first-line 伪元素与比这个例子中的选择器更复杂的选择器结合使用。例如，如果要选择包含在 project 类元素中的每个段落的第一个字母，可以使用 .project p:first-letter。

提示 只有某些特定的 CSS 属性可以应用于 :first-letter 伪元素，包括 font、color、background、text-decoration、vertical-align（只要 :first-letter 不是浮动的）、text-transform、line-height、margin、padding、border、float 和 clear。我们将在第 10 章和第 11 章讲到这些属性。

提示 第一个字母前面的标点符号（如引号）会被当做第一个字母的一部分，一同被格式化。现代浏览器都支持这一特性，但 IE8 之前的版本并不是这样做的，它将标点符号本身当做第一个字母。

伪元素、伪类及 CSS3 语法

在 CSS3 中，:first-line 的语法为 ::first-line，:first-letter 的语法为 ::first-letter。注意，它们用两个冒号代替了单个冒号。这样修改的目的是将伪元素（有四个，包括 ::first-line、::first-letter、::before 和 ::after）与伪类（如 :first-child、:link、:hover 等）区分开。

伪元素（pseudo-element）是 HTML 中并不存在的元素。例如，定义第一个字母或第一行文字时，并未在 HTML 中作相应的标记。它们是另一个元素（在本例中为 p 元素）的部分内容。

相反，**伪类**（pseudo-class）则应用于一组 HTML 元素，而你无需在 HTML 代码中用类标记它们。例如，使用 :first-child 可以选择某元素的第一个子元素，你就不用写成 class="first-child"。更多关于伪类的内容，请参见下一节。

未来，::first-line 和 ::first-letter 这样的双冒号语法是推荐的方式，现代浏览器也支持它们。原始的单冒号语法则被废弃了，但浏览器出于向后兼容的目的，仍然支持它们。不过，IE9 之前的 Internet Explorer 版本均不支持双冒号。因此，你可以选择继续使用单冒号语法，除非你为 IE8 及以下版本设置了单独的 CSS（参见 http://reference.sitepoint.com/css/conditionalcomments）。

9.7 按状态选择链接元素

CSS 允许根据链接的当前状态对它们进行格式化，参见图 9.7.1。链接的状态包括访问者是否将鼠标停留在链接上，链接是否被访问过，等等。可以通过一系列伪类实现这一特性。

```
...
    <p>Many tourists are drawn to Barcelona
    → to see Antoni Gaudí's incredible
    → architecture.</p>

    <p>Barcelona <a href="http://www.
    → gaudi2002.bcn.es/english/">
    → celebrated</a> the 150th anniversary
    → of Gaudí's birth in 2002.</p>
...
```

图 9.7.1 无法在代码中指定链接的状态。链接的状态是由访问者控制的。伪类让你可以获取链接的状态，以改变链接在该状态下显示的效果

按状态选择要格式化的链接元素的步骤

(1) 输入 a（a 是链接元素的名称）。

(2) 输入 :（冒号），前后都没有空格。

(3) 完成第 (2) 步以后，执行下列操作之一以表明你希望影响的链接状态（参见图 9.7.2）：

- ❑ 输入 link 以设置从未被激活或指向，当前也没有被激活或指向的链接的外观（如图 9.7.3 所示）；
- ❑ 输入 visited 以设置访问者已激活过的链接的外观（如图 9.7.4 所示）；
- ❑ 输入 focus，前提是链接是通过键盘选择并已准备好激活的，如图 9.7.5 所示（注意：如果链接处于活跃状态也会获得焦点）；
- ❑ 输入 hover 以设置光标指向链接时链接的外观（如图 9.7.6 所示）；
- ❑ 输入 active 以设置激活过的链接的外观（如图 9.7.7 所示）。

```
a:link {
    color: red;
}

a:visited {
    color: orange;
}

a:focus {
    color: purple;
}

a:hover {
    color: green;
}

a:active {
    color: blue;
}
```

图 9.7.2 应该始终按照这种顺序定义链接的样式，以避免链接处于多种状态（如已访问和鼠标停留）时覆盖属性

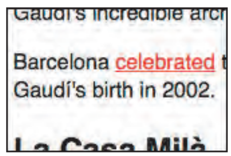


图 9.7.3 新的、未访问的链接显示为红色。另见彩插

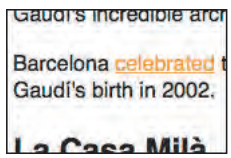


图 9.7.4 访问过的链接变为橙色。另见彩插

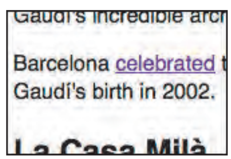


图 9.7.5 链接获得焦点（如通过 Tab 键）时显示为紫色。另见彩插

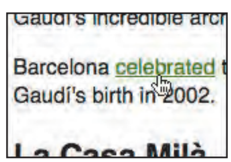


图 9.7.6 当访问者将鼠标指针停留在链接上时，它显示为绿色。另见彩插

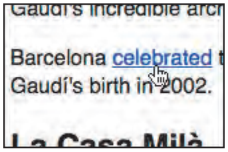


图 9.7.7　当访问者激活链接时，它变为蓝色。另见彩插

提 示　要对链接指定样式，不一定要指定伪类（因此第 (2) 步和第 (3) 步都是可选的步骤）。例如，使用 **a { color: red; }** 会让链接的所有状态显示为同一种样式。不过，最好使用伪类区分不同状态的样式，这样做可以方便访问者。

提 示　也可以对其他类型的元素使用 :active 和 :hover 伪类。例如，设置 **p:hover { color: red; }** 以后，鼠标停留在任何段落上段落都会显示为红色。

提 示　由于链接可能同时处于多种状态（如同时处于激活和鼠标停留状态），且晚出现的规则会覆盖前面出现的规则，所以，一定要按照下面的顺序定义规则：link、visited、focus、hover、active（缩写为 LVFHA）。一种助记口诀为“Lord Vader’s Former Handle Anakin”（达斯·维达的原名叫安纳金）。有人提议使用 LVHFA 的顺序，这也是可行的。

提 示　触屏设备（如智能手机和平板电脑）的浏览器没有桌面浏览器所具有的“鼠标悬停”（即 hover）状态。这是因为，像 iPad 这样的设备无法判断手指停留在一个链接上的时刻，只能判断手指轻触该链接以激活它的时刻。不过，在 iPhone 和 iPad 上，访问者激活链接时，确实会显示通过 :hover 指定的样式。其他设备的行为则不一而定。

9.8　按属性选择元素

可以对具有给定属性或属性值的元素进行格式化，参见图 9.8.1。CSS 提供了多种方式匹配属性和属性值，包括只检查属性名，检查全部或者部分属性值（更多信息请参见表 9.8.1）。如果在选择器中忽略属性值（参见图 9.8.2），就可以为具有给定属性的元素应用样式，而不管具体的属性值是什么（参见图 9.8.3）。

```
...
<article class="architect">
    <h1>Antoni Gaudí</h1>

    <p class="intro">Many tourists are
    → drawn to Barcelona to see Antoni
    → Gaudí's incredible architecture.</p>

    <p>Barcelona <a href="http://www.
    → gaudi2002.bcn.es/english/" rel=
    → "external">celebrated the 150th
    → anniversary</a> of Gaudí's birth
    → in 2002.</p>

    <h2 lang="es">La Casa Milà</h2>
    <p class="highlight">Gaudí's work was
    → essentially useful. <span lang="es">La
    → Casa Milà</span> is an apartment
    → building and <em>real people</em> live
    → there.</p>

    <h2 lang="es">La Sagrada Família</h2>
    <p>The complicatedly named and curiously
    → unfinished Expiatory Temple of the
    → Sacred Family is the <em>most visited
    → </em> building in Barcelona.</p>
</article>
...
```

图 9.8.1　出于演示目的，我为两个段落都添加了 class 属性

```
p[class] {
    color: red;
}
```

图 9.8.2　方括号包围目标属性和目标属性值。这个例子中没有属性值，它选择的是所有具有 class 属性的段落

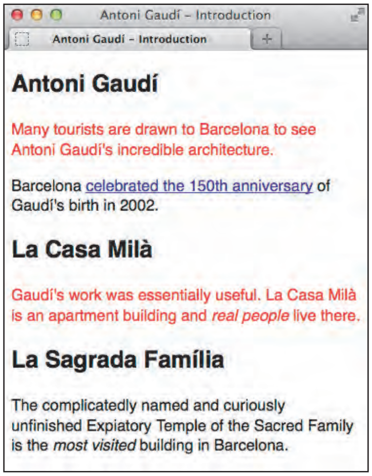


图 9.8.3　每个包含 class 属性的 p 元素（无论其 class 值是什么）都显示为红色。如果选择器是 p[class="intro"]，只有第一个段落会显示为红色。如果选择器是 p[class^="intro"]，则第一个段落以及 class="introduction" 和 class="introductory" 的段落也显示为红色

按属性选择要格式化的元素

(1) 输入 *element*，这里的 *element* 是要考察其属性的元素的选择器。

(2) 输入 *attribute*，这里的 *attribute* 是选择元素需要考察的属性的名称。

(3) 以下步骤可选，根据需要输入：

- ❑ 输入 ="*value*"，表示属性值等于这里的 *value* 的元素将被选中。
- ❑ 输入 ~="*value*"，表示属性值包含这里的 *value* 的元素将被选中（属性值还可以包含其他内容，不同的属性值之间用空格分隔）。它必须匹配完整的单词，而不是单词的一部分。
- ❑ 输入 |="*value*"（前面是管道符号，而不是数字 1 或小写字母 l），表示属性值等于这里的 *value* 或以 *value*- 开头的元素将被选中。不要输入连字符，浏览器知道搜索连字符（这常常用以搜索包含 lang 属性的元素，如在 HTML 中，[lang|="en"] 会同时匹配 lang="en" 和 lang="en-US"）。
- ❑ 输入 ^="*value*"，表明属性值以这里的 *value* 开头（作为完整的单词，或单词的一部分）的元素将被选中（这是 CSS3 中新增的特性，参见本节提示）。
- ❑ 输入 $="*value*"，表明属性值以这里的 *value* 结尾（作为完整的单词，或单词的一部分）的元素将被选中（这是 CSS3 中新增的特性，参见本节提示）。
- ❑ 输入 *="*value*"，表明属性值至少包含这里的 *value* 一次的元素将被选中。也就是说，*value* 不必是属性值中的完整单词（这是 CSS3 中新增的特性，参见本节提示）。

(4) 输入]。如果你想为元素指定其他属性或者属性值，可重复第 (2) ~ (4) 步。

表 9.8.1　属性选择器参考表

选　择　器	属　性　值
[attribute]	匹配指定属性，不论具体值是什么
[attribute="value"]	完全匹配指定属性值
[attribute~="value"]	属性值是以空格分隔的多个单词，其中有一个完全匹配指定值
[attribute|="value"]	属性值以 value- 打头
[attribute^="value"]	属性值以 value 开头，value 为完整的单词或单词的一部分
[attribute$="value"]	属性值以 value 结尾，value 为完整的单词或单词的一部分
[attribute*="value"]	属性值为指定值的子字符串

提 示　当前所有主流浏览器均支持按元素包含的属性（和属性值）选择元素。对于第 (3) 步中提到的 CSS3 中新增的属性选择器，IE7 和 IE8 有一些异常。更多信息参见 http://reference.sitepoint.com/css/css3attributeselectors。

更多属性选择器示例

属性选择器看起来都长得差不多（参见表 9.8.1），但是它们使用起来非常灵活，可以以多种方式匹配元素的属性及其值。这里给出更多的例子，演示属性选择器几种不同的使用方式。这些例子在实践中也用得上。

- 这个选择器选择任何 rel 属性值等于 external（必须完全匹配）的 a 元素。对任何链接到站外页面的 a 元素（即外部链接）添加 rel="external" 是非常好的习惯。使用下面的样式规则，可为外部链接添加独特的样式，从而告诉访问者点击该链接将离开你的网站。

```
a[rel="external"] {
     color: red;
}
```

- 假设某 article 元素有两个类，如 <article **class="project barcelona"**>，另一个 article 有一个类，如 <article **class="barcelona"**>。~= 语法可以测试单词的部分匹配，即匹配以空格相隔的多个单词中的一个。在这个例子中，两个元素都将显示为红色。

```
article[class~="barcelona"] {
     color: red;
}

/* 这个选择器也能匹配，因为这个选择器匹配部分字符串（不需要完整的单词） */
   article[class*="barc"] {
     color: red;
}

/* 这个选择器无法匹配，因为barc并不是空格分隔的单词列表中的某个完整的单词 */
   article[class~="barc"] {
     color: red;
}
```

- 这个选择器选择任何带有 lang 属性且属性值以 es 开头的 h2（如 es-ES 和 es-PE，前者代表西班牙西班牙语，后者代表秘鲁西班牙语）。在 HTML 代码示例（参见图 9.8.1）中有两个这样的实例。

```
h2[lang|="es"] {
    color: red;
}
```

- 通过使用通用选择器，这个选择器选择任何带有 lang 属性且属性值以 es 开头的元素。在 HTML 代码示例（参见图 9.8.1）中有三个这样的实例。

```
*[lang|="es"] {
    color: red;
}
```

- 通过联合使用多种方法，这个选择器选择所有既有任意 href 属性，又有任意属性值包含单词 howdy 的 title 属性的 a 元素。

```
a[href][title~="howdy"] {
    color: red;
}
```

- 作为上一选择器的精确度低一些的变体，这个选择器选择所有既有任意 href 属性，又有任意属性值包含 how（作为完整的单词或单词的一部分，它匹配 how、howdy、show 等，无论 how 出现在属性值的什么位置）的 title 属性的 a 元素。

```
a[href][title*="how"] {
    color: red;
}
```

- 这个选择器匹配任何 href 属性值以 http:// 开头的 a 元素。

```
a[href^="http://"] {
    color: orange;
}
```

- 这个选择器匹配任何 src 属性值完全等于 logo.png 的 img 元素。

```
img[src="logo.png"] {
    border: 1px solid green;
}
```

- 这个选择器的精确度比前一个低一些，它匹配任何 src 属性值以 .png 结尾的 img 元素。

```
img[src$=".png"] {
    border: 1px solid green;
}
```

只用上面这些例子很难全面概括使用选择器能够实现的效果，不过，希望它们可以激发你继续探索。

9

9.9　指定元素组

我们经常需要将相同的样式规则应用于多个元素。可以为每个元素重复地设置样式规则，也可以组合选择器，一次性地设置样式规则（参见图 9.9.1 至图 9.9.3）。当然，后一种方法效率更高，通常也会让样式表更易于维护。

```
…
<article class="architect">
    <h1>Antoni Gaudí</h1>
    <p>Many tourists are drawn …</p>
    <p>Barcelona …</p>

    <h2 lang="es">La Casa Milà</h2>
    <p>Gaudí's work was …</p>

    <h2 lang="es">La Sagrada Família</h2>
    <p>The complicatedly named …</p>
</article>
…
```

图 9.9.1　这段代码包含一个 h1 和两个 h2

```
h1,
h2 {
    color: red;
}
```

图 9.9.2　可以列出任意数量的单独的选择器（无论它们包含的是元素名称、类还是伪元素），只需用逗号分隔它们即可

将样式应用于元素组的步骤

(1) 输入 *selector1*，这里的 *selector1* 是受样式规则影响的第一个元素的名称。

(2) 输入 **,**（逗号）。

(3) 输入 *selector2*，这里的 *selector2* 是受样式规则影响的下一个元素的名称。

(4) 对其他每个元素重复第 (2) 步和第 (3) 步。

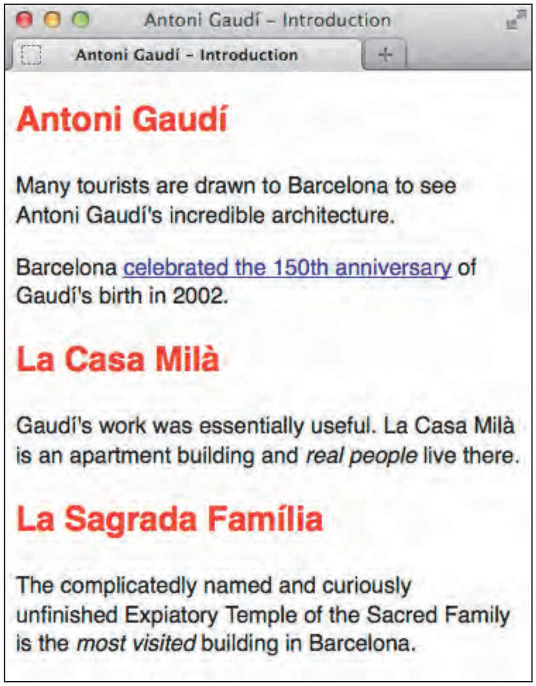


图 9.9.3　通过同一条样式规则，可以让 h1 和 h2 元素都显示为红色

提示　通过元素组添加样式只是一种简写方式。图 9.9.2 中的规则跟 **h1 { color: red; }** 和 **h2 { color: red; }** 这两个规则的效果完全一样。

提示　可以组合使用任何类型的选择器，从最简单的（如图 9.9.2 所示）到最复杂的都可以。例如，可以使用 h2, .project p:first-letter 来选择二级标题以及包含在 class 等于 project 的所有元素中的 p 元素的第一个字母。

提示　不同的选择器不一定非得单独成行（图 9.9.2 采用的就是这种做法），但很多编码人员都遵循这种惯例，易于阅读！

提 示 有时，为应用于多个选择器的共同样式创建一个样式规则，再为没有共同点的样式分别创建单独的样式规则是很有用的。要记住，在样式表中后指定的规则会覆盖先指定的规则。

9.10 组合使用选择器

本章的示例都尽量保持简单，以便你全面了解各种类型的选择器。不过，现实中常常需要组合使用这些技术，才能找到要格式化的元素，但这也是选择器的功能强大之所在。

图 9.10.1（实现结果见图 9.10.2）通过一个极端的例子展示了如何组合使用选择器，不过此处我们不推荐采用这种方法。这里展示了可实现相同效果的几种方式，它们是按特殊性由低向高排列的。

```
em {
    color: red;
}

.project em {
    color: red;
}

.architect .project em {
    color: red;
}
```

没有必要搞一大堆吓唬人的选择器来实现网页中的大部分设计，不管这些设计在浏览器中看起来多么错综复杂。只在必要情况下组合使用选择器，并且，最好将特殊性控制在刚好需要的程度。例如，如果你只想选择包含在 class="project" 的元素内的 em 元素，可以使用 **.project em { color: red; }**。尽管在 HTML 中 em 元素是嵌套在 p 元素中的，但没有必要写成 .project p em { color: red; }，除非你不想为段落**以外**的 em 元素应用该样式。总之，从最简单的开始，按需增加特殊性。

```
.project h2[lang|="es"] + p em {
    color: red;
}
```

图 9.10.1 这对你来说可能是个挑战。从右向左看，它表明“仅选择 em 元素，它们包含在 p 元素中，这样的 p 元素是 lang 属性值以 es 开头的 h2 元素的直接相邻同胞元素，且是 class 等于 project 的任何元素的子元素”。弄明白了吗？实际上，很少需要编写这么复杂的选择器，但至少这样做是可行的



图 9.10.2 图 9.10.1 中所有那些东西只是让 em 元素显示为红色吗？如果你认为简单地写成 .em { color: red; } 或 者 .architect em { color: red; } 要好得多（也更易于维护），你肯定是对的。除非需要更高的特殊性，否则越简单越好

更多 CSS3 选择器

CSS3 为你的工具箱增加了不少新的选择器。你已经在本章中见到了其中的一些。其他的新选择器大多为伪类，其中的一些还相当复杂，不过也非常强大。所有 CSS3 选择器及其完整描述见 www.w3.org/TR/css3-selectors/#selectors，简介与示例参见 www.w3.org/wiki/CSS/Selectors。

除了 Internet Explorer 以外，其他浏览器对选择器的支持程度都很好。Internet Explorer 直到 IE9 才开始支持大多数 CSS3 中新增的选择器（尤其是伪类和伪元素）。

在很多情况下，如果你的网站在旧的浏览器里的样式与在现代浏览器里的样式稍有差别也是没有问题的。不过，如果你确实希望 IE8 甚至更旧的浏览器也能使用这些选择器，可以考虑使用 Keith Clark 的 Selectivizr（http://selectivizr.com）。根据他的介绍，这是“一个在 Internet Explorer 6 ~ 8 中模拟 CSS3 伪类和属性选择器的 JavaScript 工具”。根据他的提示，Selectivizr 有一些限制。我的建议是尽可能不使用该工具（我无意冒犯该工具本身）。页面上使用的 JavaScript 越多，页面的加载时间就越长（参见第 19 章），何况旧的浏览器（如 IE8）处理 JavaScript 的效率要低得多。

第 10 章

为文本添加样式

本章内容

- 本章之前与本章之后
- 选择字体系列
- 指定替代字体
- 创建斜体
- 应用粗体格式
- 设置字体大小
- 设置行高
- 同时设置所有字体值
- 设置颜色
- 设置背景
- 控制间距
- 增加缩进
- 对齐文本
- 修改文本的大小写
- 使用小型大写字母
- 装饰文本
- 设置空白属性

使用 CSS 可以修改文本的字体、大小、粗细、倾斜、行高、前景和背景颜色、间距和对齐方式，可以决定是否为文本添加下划线或删除线，可以将文本转化为全部使用大写字母、全部使用小写字母或使用小型大写字母。而且，通过短短几行代码就可以让这些样式应用于整篇文档或整个网站。在本章中，你将学习如何做到这些。

本章讨论的很多属性只是应用到了文本，10.10 节中讨论的属性也应用到了非文本元素。另外其他章节也涉及一些文本相关主题：第 13 章讲到了 Web 字体，第 14 章讲到了为文本添加阴影。

所有这些 CSS 特性都是设计网页不可或缺的部分。在第 11 章讲解 CSS 布局时我们还会继续关注 CSS 特性。

10.1 本章之前与本章之后

浏览器会给页面添加极少量的默认样式（如图 10.1.1 所示）。在本章中，我们将尝试对文本和背景应用我们自己的 CSS，从而改善页面的外观（如图 10.1.2 所示）。在学习本章的过程中，你可能需要用到图 10.1.3 所示的简化过的 HTML（尤其要注意其中的类名是如何在 CSS 中使用的）。完整的 HTML 及本章的所有示例都可以在本书配套网站上查找，见 www.htmlcssvqs.com/8ed/10。



图 10.1.1　页面的默认样式在所有的浏览器中都是与此类似的（但标题的字号大小在不同的浏览器中可能略有差异）



图 10.1.2　你即将了解到，对页面施加一系列样式可以改变页面的外观。可以通过 www.htmlcssvqs.com/8ed/text-final 访问这个页面

```
...
<body>
<article class="architect">
    <div class="intro">
       <h1>Barcelona's Architect</h1>

       <p class="subhead">Antoni Gaudí's
       → incredible buildings bring...</p>

       <p>Gaudí's...search for simplicity...is
       → quite apparent in his work, from the
       → <a href="#park-guell">Park Guell</a>
       → ... to the Church of the <a href=
       → "#sagrada-familia">Sacred Family
       → </a> ...</p>
    </div>

    <section class="project family">
       <h2 id="sagrada-familia">La Sagrada
       → Família</h2>

       <div class="photos">
          <img src="img/red-bubbles.jpg" ... />
          ... 6 more images ...
       </div>

       <p>The complicatedly named...</p>
       <p>The Sagrada Família attracts...</p>
    </section>

    <section class="project guell">
       <h2 id="park-guell">Park Guell</h2>

       <div class="photos">
          ... 5 images ...
       </div>

       <p>The Park Guell always reminds me
       → of ... Howard Roark in ... <a href=
       → "http://..."><cite>The Fountainhead
       →  </cite></a>...</p>
       <p>...now we <em>all</em> get to enjoy
       → it...</p>
    </section>
</article>
</body>
</html>
```

图 10.1.3　HTML 中包含了很多类名，方便 CSS 找到这些特殊的区域

10.2　选择字体系列

对于自己的网站，一大关键问题就是选择标题和主体文本所用的字体。你将了解到，并非所有的系统都支持相同的默认字体，因此，应该定义替代字体作为备选。不过，让我们首先看看如何定义单个字体（如图 10.2.1 和图 10.2.2 所示），以及未提供替代字体的影响（如图 10.2.3 所示）。

设置字体的方法

在样式表中需要的选择器之后，输入 `font-family: name`，这里的 *name* 是首选字体的名称，参见图 10.2.1。

```
body {
    font-family: Geneva;
}

h1, h2 {
    font-family: "Gill Sans";
}
```

图 10.2.1　由于 `font-family` 是继承属性，而我们对 body 元素设置了 Geneva 字体，该样式会应用到其他元素。通过对 `h1` 和 `h2` 元素设置 Gill Sans 字体，覆盖了继承属性



图 10.2.2　在 OS X 系统中，Geneva 和 Gill Sans 很常见，因此可以正常显示。`p` 元素和 `a` 元素继承了 `body` 的 `font-family` 设置，除了标题之外的文本都显示为 Geneva。如果没有为 `h1` 和 `h2` 指定 Gill Sans 字体，它们也将显示为 Geneva 字体



图 10.2.3　一些 Windows 系统中并未安装 Geneva 和 Gill Sans 字体。如果选择了访问者系统中没有的字体，他们的浏览器中就会显示默认字体（如图所示，Windows 中的默认字体是 Times New Roman）

提 示　对于包含多个单词的字体名称，应该用引号（单引号或双引号）包围起来。

提 示　可以使用小写字母指定字体名称，如 `font-family: geneva;`。

提 示　开发人员可以指定自己想要设置的任何字体，不过访问者只会看到他们的系统里安装的字体（一种特殊情况是加载 Web 字体，参见第 13 章）。关于 Windows 和 Mac OS X 共享的标准字体的更多信息参见下一节。

提 示　虽然 `font-family` 属性是继承的，但有几个元素不会继承父元素的字体设置，其中有表单的 `select`、`textarea` 和 `input` 元素（参见第 16 章）。不过，可以强制它们继承父元素的字体设置，代码为 `input, select, textarea { font-family: inherit; }`。

提 示　使用通用的 `font` 属性可以一次性定义字体、大小和行高。参见 10.8 节。

10.3　指定替代字体

尽管你可以指定自己想要的任何字体，但这些字体不一定在每个系统上都能显示出来（如上一节所示）。为此，可以在 `font-family` 属性中列举一种以上的字体（如图 10.3.1 所示）。浏览器会使用列表中第一个已安装在访问者系统内的字体（如图 10.3.2 所示和图 10.3.3 所示）或 Web 字体（参见第 13 章）。

```
body {
    font-family: Geneva, Tahoma, sans-serif;
}

h1,
h2 {
    font-family: "Gill Sans", "Gill Sans MT",
    ➝ Calibri, sans-serif;
}
```

图 10.3.1　第一字体栈告诉浏览器如果系统上没有 Geneva 就用 Tahoma，如果这两个都没有就用标准的 sans-serif。标题的字体栈提供了三种备选字体，替换字体可能无法完全匹配首先字体，不过，我们的目标是指定尽可能接近的字体



图 10.3.2　安装了 Geneva 和 Gill Sans 的系统（如这里的 OS X）会继续使用该字体。因此，这里显示的字体与图 10.2.2 是完全一样的（截图看起来不一样，这是因为本图里的浏览器宽度更窄）

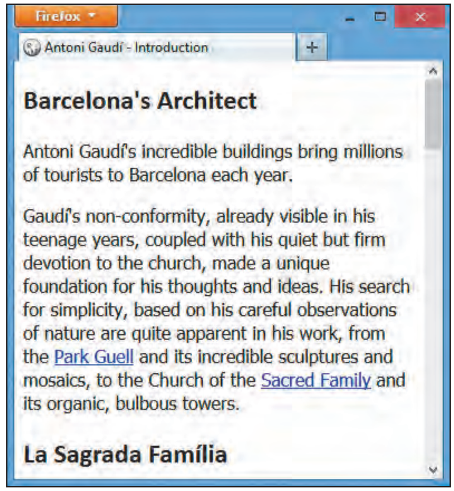


图 10.3.3　没有 Geneva 字体的系统会使用 Tahoma 字体（对于这种情况，大多数 Windows 系统都会使用后一种字体）。如果 Tahoma 也没有，浏览器会使用 sans-serif 字体。标题字体也用同样的方法处理。Windows 机器既没有 Gill Sans，也没有 Gill Sans MT，标题是使用第三种替换字体 Calibri 显示的

字体列表称为**字体栈**（font stack）。通常，字体栈至少包含三个字体：希望使用的字体、一个或几个替代字体，以及一个表示类属的标准字体，表示“如果其他的字体都不可用，就用这个”（参见第一条提示）。

补充材料“OS X 和 Windows 上默认共有的字体”讨论了两个系统共享的几种字体。使用两种系统上的共享字体或者 Web 字体的情况很常见，因此通常访问者看到的字体都是一样。

指定替代字体

(1) 输入 `font-family:` *`name`*，这里的 *`name`* 是首选字体的名称。

(2) 输入 `,` *`name2`*，这里的 *`name2`* 是第二字体的名称。用一个逗号和空格分隔每个字体。

(3) 根据需要，重复第 (2) 步，最后以一个表示类属的标准字体（`serif`、`sans-`

serif、cursive、fantasy 或 monospace[①]，它们代表首选字体最为接近的风格）结束字体列表。

OS X 和 Windows 上默认共有的字体

OS X 和 Windows 上默认都有的字体是非常有限的，它们仅包括 Arial、Comic Sans MS、Courier New、Georgia、Impact、Trebuchet MS、Times New Roman 和 Verdana。结果，绝大多数网站使用的字体都无外乎这些。尽管它们在 Mac OS 和 Windows 上的浏览器中显示的效果也并不完全一致，但至少你可以肯定它们都能显示出来。

除此之外，还有别的选择（如图 10.3.1 所示）。OS X 和 Windows 都包含了更多（却不相同的）可用于字体栈的系统字体。在网上搜索“font stacks”（字体栈），就可以看到很多 font-family 声明。可以将这些声明复制到你的样式表里，从而为不同的访问者提供相似的字体。

此外，还可以在网页中加载系统默认没有的字体。Web 设计人员多年来都痴迷于这种想法，近年来终于成为现实。Web 字体变得越来越普遍，第 13 章将讲解如何使用 Web 字体。

在 Windows 上显示 Arial，在 OS X 上显示 Helvetica

这里将演示一个小技巧。不过在此之前，先介绍一些背景知识。

几乎可以确定 Arial 是万维网上用的最多的字体，因为 font-family: arial, helvetica, sans-serif; 这条声明可谓随处可见。问题是，设计师通常更愿意在能使用 Helvetica 的情况下使用 Helvetica，但大多数 OS X 机器也安装有 Arial，因此在这个字体栈中，Helvetica 就被忽略了。

所有的 OS X 机器都有 Helvetica，但 Windows 上一般没有这个字体。即便哪个用户在 Windows 上安装了这个字体，其显示效果也不太如意。这是字体栈中不会将 Helvetica 列在 Arial 前面的原因。

怎么办呢？解决方案很简单，就是使用 font-family: sans-serif;。这种方法会让所有的操作系统使用无衬线字体——Windows 的浏览器显示 Arial，而 OS X 的浏览器则显示 Helvetica。

提 示 系统通常都具有下列表示类属的字体名称对应的字体：serif、sans-serif、cursive、fantasy 和 monospace。因此，标准的做法是在字体栈的末尾指定上述字体名称中的一种。到目前为止，serif 和 sans-serif 是用得最多的，因为它们分别对应于最为常见的两类字体。serif 通常对应 Windows 上的 Times New Roman 和 OS X 上的 Times。sans-serif 通常对应 Windows 上的 Arial 和 OS X 上的 Helvetica。

① cursive、fantasy、monospace 分别表示手写字体、装饰字体和等宽字体。——译者注

提 示 Geneva 字体栈通常会包含 Verdana，将其作为第三个选项，以应对既没有 Geneva 也没有 Tahoma 的少量操作系统。这里没有包含该字体，主要是为了演示字体栈可以包含不同数量的字体，不过在接下来的例子中，会将它放进字体栈里。

提 示 可以在同一个 font-family 规则中为不同的字母表指定字体（如日语和英语），从而对包含不同语言和书写体系的文本进行格式化。

10.4 创建斜体

在传统出版业中，斜体通常用来表示引述、强调的文本、外文单词（如 *de rigueur*）、学名（如 *Homo sapiens*）、电影片名等。

浏览器通常让一些 HTML 元素（如 cite、em 和 i）默认以斜体显示，因此，不必在 CSS 中对这些元素设置斜体。有时你需要让一些内容以斜体显示，但上述元素从语义上来说都不合适。使用 CSS 的 font-style 属性可以让任何元素中的文本以斜体显示。

作为示例，图 10.4.1 说明了如何对段落文本添加斜体样式。（由于这样显示阅读体验太差，如图 10.4.2 所示，因此，接下来的例子中不会保留这一样式规则。）

```
body {
    font-family: Geneva, Tahoma, Verdana,
    ➝ sans-serif;
}

h1,
h2 {
    font-family: "Gill Sans", "Gill Sans MT",
    ➝ Calibri, sans-serif;
}

p {
    font-style: italic;
}
```

图 10.4.1 在这个例子中，段落以斜体显示



图 10.4.2 两个标题以正体显示，它们之间的段落以斜体显示

1. 创建斜体

(1) 输入 font-style:。

(2) 在 :（冒号）后输入 italic（创建斜体文本）或 oblique（创建倾斜文本）。（99% 的情况下都会使用 italic。在所有情况下，很难察觉使用 oblique 与 italic 的差异。）

2. 取消斜体

输入 font-style: normal。

提 示 想取消斜体的一个可能的原因是，要在从父元素中继承了斜体格式的段落中对某些文字进行强调。关于继承的更多细节，参考 7.3 节。

提示 关于编写语义化 HTML 的重要性参见 1.8 节。

提示 font-style 属性是继承的。

真斜体、假斜体、倾斜版本的区别

字体的斜体版本通常是由字体设计师从头创建的，尤其是有衬线字体。字体的斜体版本不仅仅是普通文本的倾斜版本，它们的形式有一些合理的差异。例如，Palatino Linotype 就有一份真正的斜体字体，参见图 10.4.3。通过字母 a 可以很清晰地看到，斜体不仅仅是对字母进行倾斜产生的效果。

不过，有的字体并没有斜体版本。如果对这些字体的文本设置 font-style: italic，浏览器就会显示一种计算机模拟出来的假斜体，即简单地对普通字母进行倾斜以模拟斜体风格（参见图 10.4.3）。这两种方式的质量是有所不同的。

此外，字体设计师还可能专门为字体创建倾斜版本，它们通常是对普通字母进行倾斜，同时可能对字符间距等进行微调而产生的，与普通字母相比，字母本身的形状是相同的。要使用字体的倾斜版本，可以设置 font-style: oblique;，不过这样做并不常见。如果字体没有斜体版本或倾斜版本，就会以伪斜体显示。



图 10.4.3 这里有两组应用了不同字体的文字，每组的第一行都是普通的文本，第二行是添加了 font-style: italic; 的文本。如果仔细观察，你会发现，Palatino Linotype 字体对斜体字母作了特殊处理。正体与斜体的区别在 a、p 和 y 等字母上表现得最为明显。由于 Geneva 没有斜体字母，因此浏览器只是简单地让普通文本倾斜，模拟斜体效果

10.5 应用粗体格式

粗体格式可能是让文本突出显示的最常见、最有效的方式。例如，浏览器通常默认为 h1 ~ h6 添加粗体格式。同斜体一样，可以为任何格式添加或取消粗体。添加粗体和取消粗体使用 font-weight 属性（参见图 10.5.1 和图 10.5.2）。

1. 应用粗体格式

(1) 输入 font-weight:。

(2) 输入 bold，让文本显示为具有平均加粗值的粗体。这一属性值适用于大多数情况。

或者输入 100 ~ 900 之间的 100 的倍数，其中 400 代表正常粗细，700 代表粗体。如果使用的是具有多种粗细版本的字体，这种方法就很有用（有些 Web 字体就采用了这种方法）。

或者输入 bolder（更粗）或 lighter（更细）以设置相对当前粗细的值。

10

```
body {
    font-family: Geneva, Tahoma, Verdana,
    → sans-serif;
}

h1,
h2 {
    font-family: "Gill Sans", "Gill Sans MT",
    → Calibri, sans-serif;
    font-weight: normal;
}

em,
a:link,
.intro .subhead {
    font-weight: bold;
}
```

图 10.5.1　浏览器会自动为标题 h1 ~ h6 添加粗体格式。我为所有的 h1 和 h2 元素应用了正常的粗细，取消了自动添加的粗体格式。同时，我为 em 文本及链接、.subhead 添加了粗体格式（参见图 10.5.2）

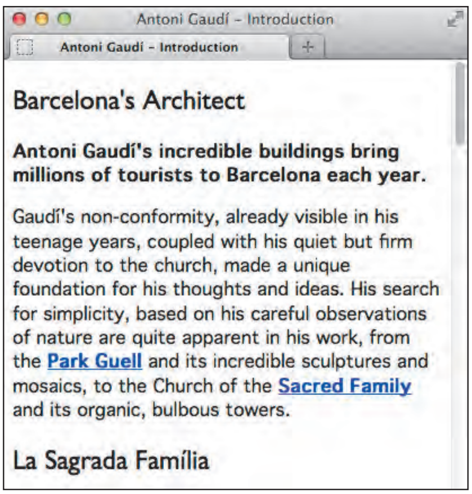


图 10.5.2　标题是以常规粗细显示的，而不是默认的粗体（参见图 10.4.2）。.subhead 类段落是以粗体显示的，链接也是

2. 取消粗体格式

输入 font-weight: normal。（有些 Web 字体服务，如 Google Fonts，要求使用 font-weight: 400。）

提 示　由于各种字体定义粗细的方式并不相同，因此预定义值在不同字体上的表现可能不一致。预定义值用以表示某一给定字体下的相对值，如 700 在一种字体中看起来可能比在另一种字体中要粗。

提 示　字体本身有时候不包括与相关值对应的不同程度的粗细。如果字体的粗细少于九种，或者这些粗细集中于范围的一端，那么一些数值就可能对应于同样的粗细，在网页上的显示效果就都一样，参见图 10.5.3。

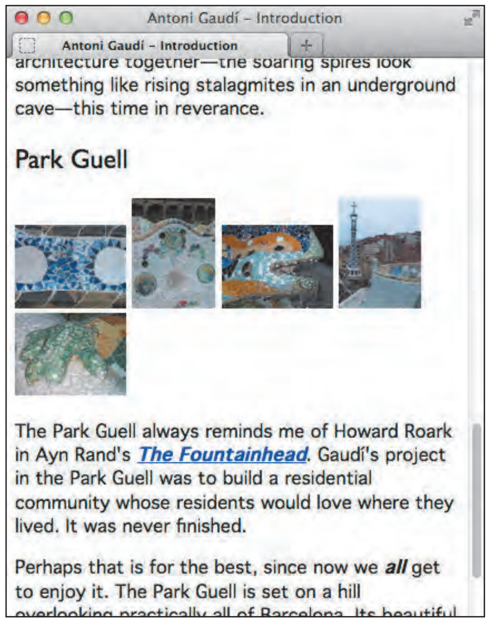


图 10.5.3　在页面的底部，可以看到一个链接（The Fountainhead）和单词 all。根据我们新添加的样式，它们都以粗体显示，同时它们还会因为浏览器的默认样式而显示为斜体。根据其含义这两个短语分别由 cite 和 em 标记（注意 h2 文字 Park Guell 是普通字体，没有加粗）

提 示　鉴于前两条提示，添加粗体样式的通行做法是简单地写成 font-weight: bold。

提 示 有的字体没有设计粗体字母。对于这些字体，如果声明 font-weight: bold;，浏览器会显示伪粗体（如图 10.5.4 和图 10.5.5 所示）。对伪斜体的讨论参见 10.4 节。

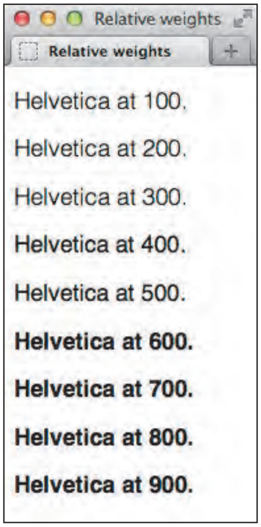


图 10.5.4 有的粗细值不同的文本显示的效果一致，是因为 Helvetica 并没有为每种粗细值都建立一套字母集

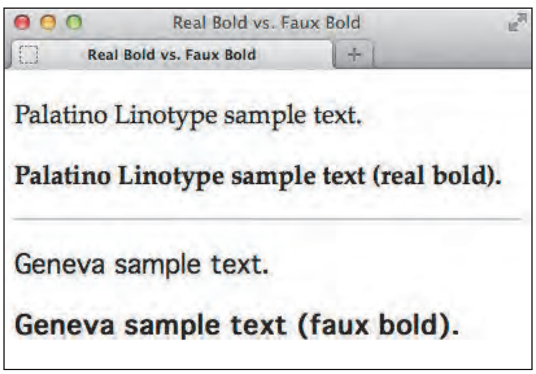


图 10.5.5 同斜体类似，Palatino Linotype 字体有粗体字母，而 Geneva 没有。因此，对于 Geneva，浏览器会将字母加粗，模拟粗体效果，而这样做的结果则是不够锐利、优雅，字母间隔也不是特别理想

提 示 哪些元素是有可能需要取消粗体的呢？这包括任何自动添加了粗体格式的元素（应该能想到 strong、h1 ~ h6 和 b），以及继承了父元素粗体格式的元素（参见 7.3 节）。

提 示 font-weight 属性是继承的。

10.6 设置字体大小

为网页里的文本设置字体大小有两种方式：直接使用像素指定要使用的特定字号（如图 10.6.1 ~ 10.6.3 所示），或使用百分数、em（如图 10.6.4 和图 10.6.5 所示）或者 rem 指定相对于父元素文本的大小。关于 rem，参见补充材料“使用 rem 指定字体大小”。

```
... [previous CSS] ...

em,
a:link,
.intro .subhead {
     font-weight: bold;
}

h1 {
     font-size: 35px;
}

h2 {
     font-size: 28px;
}

.intro .subhead {
     font-size: 18px;
}

.intro p {
     font-size: 17px;
}

.project p {
     font-size: 15px;
}
```

图 10.6.1 这里使用像素值指定文本大小（效果参见图 10.6.2 和图 10.6.3）



图 10.6.2　图 10.6.1 中指定的字体在浏览器中的显示效果。标题字号的大小显示了信息的层级关系。主标题下面的介绍区域中的段落文本字号稍大

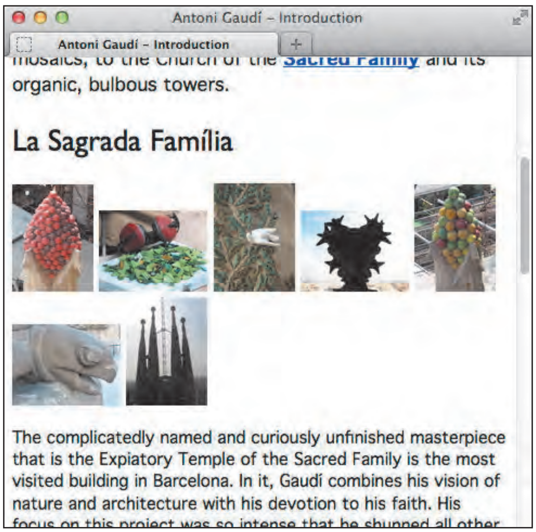


图 10.6.3　页面中剩余部分的段落文本

对初学者来说，像素更容易理解，不过强烈建议你一开始就学习使用 em 这样的相对单位。使用相对单位可以有更大的灵活性，而且对定义页面中特定的设计部件（如空白、边距等）的尺寸很有帮助。在各种尺寸的设备（智能手机、平板电脑等）不断涌现的今天，使用相对单位有助于建立在各种设备都能显示良好的页面（这就是响应式 Web 设计涉及的内容，参见第 12 章）。

```
body {
     font-family: Geneva, Tahoma, Verdana,
     ➝ sans-serif;
     font-size: 100%; /* 16px */
}

h1,
h2 {
     font-family: "Gill Sans", "Gill Sans MT",
     ➝ Calibri, sans-serif;
     font-weight: normal;
}

h1 {
     font-size: 2.1875em; /* 35px/16px */
}

h2 {
     font-size: 1.75em; /* 28px/16px */
}

em,
a:link,
.intro .subhead {
     font-weight: bold;
}

.intro .subhead {
     font-size: 1.125em; /* 18px/16px */
}

.intro p {
     font-size: 1.0625em; /* 17px/16px */
}

.project p {
     font-size: .9375em; /* 15px/16px */
}
```

图 10.6.4　body 里的 font-size: 100% 声明为 em 字体大小设置了参考的基准。这里的 100% 将被翻译为默认字体大小（大多数系统下为 16px）。这样，样式表的结果同图 10.6.1 所示的结果便是一样的。每个 font-size 属性值后面的注释解释了该值的计算方法，同时显示了等价的像素值

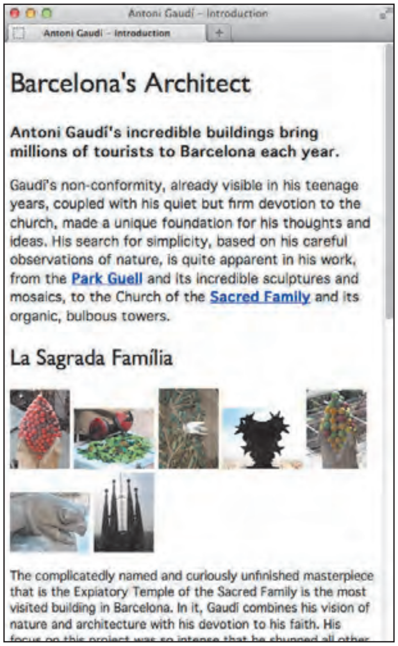


图 10.6.5　在大多数系统中，如果使用默认设置，以 em 为基础的字体大小同以像素为基础的版本是一样的（如图 10.6.2 和图 10.6.3 所示）

1. 理解 em 和百分数字体大小

设置相对于父元素的字体大小需要一点时间去习惯。你需要理解浏览器处理这些相对于父元素的单位的方式，下面花一点时间解释一下。

不过首先要说明，使用这种方法时，最好在 `body` 元素上建立一个基准，即声明 `body { font-size:100%; }`（参见图 10.6.4）。大多数时候，这项设置等价于将字体大小设为 16px，即大多数系统的默认字体大小。照例，该属性值传递至其他的元素（记住，`font-size` 是继承性属性），除非其他元素通过浏览器默认设置了自己的 `font-size`，或者在样式表指定了自己的 `font-size`。（多数元素是 16px，但默认情况下 `h1` ~ `h6` 稍大。）

那么，如何计算要指定的 em[①] 值呢？实际上，1em 总是等于默认的字号大小，这是 em 的工作原理。在这里，1em 就等于 16px，因为我们将其创建为元素的默认字号。据此可以通过一点点除法确定 em 值（或百分比）。

要指定的字体大小／父元素的字体大小＝值

让我们来看看如何将图 10.6.1 中的像素值改为 em。例如，如果要将 `h1` 设为 35px，同时已知其父元素的字体大小为 16px，因此

```
35 / 16 = 2.1875
```

因此，设置 `h1{font-size: 2.1875em;}` 就可以了（参见图 10.6.4）。这条规则表示“将 `h1` 文本的大小设置为其父元素文本大小的 2.1875 倍”。另一种方式是写成 `h1 { font-size: 218.75%; }`。不过，除了在 `body` 中设置基准 `font-size` 时使用百分数以外，设置字体大小时使用 em 比百分数更为常见。

再如，要将 `project` 类段落文本设置为 15px，而

```
15 / 16 = .9375
```

因此，设置 `p { font-size: 0.9375em; }`，参见图 10.6.4（这个值也可以设成 93.75%）。

再讨论一个例子。这可能是你感到困惑的地方。页面中的第二个段落 Gaudí’s non-conformity... 包含两个链接（如图 10.6.5 和图 10.6.6 所示）。段落本身的字体样式设置为 `.intro p { font-size: 1.0625em; }`，由于 17/16=1.0625，因而段落文本字号为 17 像素。

假设要让链接显示为 16px，同时让段落里的其他文本仍显示为 17px。你可能打算将链接的 `font-size` 设为 1em（`.intro a { font-size: 1em; }`），因为 1em = 16px。

① 在排版领域，em 是一种量度单位，1em 等于当前指定的字号大小。em 这一名称与字母 M 有关，起初，1em 等于给定字体环境下字母 M 的宽度。——译者注

```
...
<div class="intro">
    <h1>Barcelona's Architect</h1>
    ...
    <p>Gaudí's non-conformity...His search for simplicity...is quite apparent in his work, from
    → the <a href="#park-guell">Park Guell</a> ... to the Church of the <a href="#sagrada-
    → familia">Sacred Family</a> and its organic, bulbous towers.</p>
</div>
...
```

图 10.6.6 这段 HTML 包含两个 a 元素，它们都位于一个父元素 p 中。div 容器的 intro 类是为了在不影响页面中其他元素样式的情况下，给该 div 中的段落和链接设置样式而引入的（如图 10.6.1 至图 10.6.4 所示）

但需要记住的是，该值应该是相对于这些元素的父元素的。在这个例子中，它们的父元素是 p。段落的字体大小是 17px，而不是 16px，因此要让 1em = 17px，前面说过，1em 应该始终等于父元素的字体大小。

因此，要让链接显示为 16px，需要设置一个比 1em 大的 em 值：

```
16 / 17 = .941176
```

因此，使用这个有点冗长的 `.intro a { fontsize: 0.941176em; }` 将得到想要的结果（数字太长了不好看，因此小数点后我省略了几位）。

最后需要指出的是，在大多数情况下，100% 的 body font-size 等于 16px。不过有一种例外的情况，即用户对浏览器的设置覆盖了该默认值。例如，如果用户是视障人士，他们可能将默认字体大小设置为 22px。将 body 设置为 100%，页面就会以此为基准设置其余的文本的相对大小。这是使用 em 和百分数设置字体大小的美之所在。（如果你将 html 元素设置为 100%，同样的道理也适用于 rem。）参见最后一条提示，了解使用 em 的更多原因。

使用 rem 设置字体大小

CSS3 引入了一些新的单位，其中很有意思的一个便是 rem（root em 的简称）。它同 em 很像，不过它总是以根元素为参照系设置其他元素的字体大小，而不是父元素，参见图 10.6.7。根元素是 html 元素（因为它包含 body，也就包含了所有内容）。

这样做简化了字体大小的设置，因为 html 的字体大小通常不会变，不像父元素的大小是不确定的，就像在段落中设置链接字体大小的例子那样。（参见“理解 em 和百分数字体大小”部分。）因此，我们的公式（在前面公式的基础上调整而来）为：

要指定的字体大小 / 根元素字体大小 = 值

实际上就是

要指定的字体大小 / 16 = 值

是吧？我已经能听到你的欢呼了。

先别着急，虽然现代浏览器对它的支持程度很高，但 Internet Explorer 直到 IE9 才开始支持它（http://caniuse.com/#search=rem）。在世界上很多地方，IE8 仍然占据较大的市场份额，建站时不得不考虑对其兼容。这也是很多代码编写人员尚未使用 rem 的原因。

不过，有一种策略就是针对旧版本的 IE 提供以 px 为单位的值，再紧跟着为现代浏览器准备的 rem 值（如图 10.6.7 所示）。当然，这样做的一个缺点是需要维护的代码量和浏览器需要加载的代码量变多了。另一个缺点是，在 IE8 中如果文本的字体大小是以像素为单位的，用户就无法改变文字的大小。诚然，这会影响一小部分用户。请你自己权衡并作出取舍。

```
html {
    font-size: 100%; /* 通常是16px */
}

.intro p {
    font-size: 17px; /* 可选 */
    font-size: 1.0625rem; /* 17px/16px */
}

.intro a {
    font-size: 16px; /* 可选 */
    font-size: 1rem; /* 是 .941176em */
}
```

图 10.6.7 使用 rem，则只用关心根元素的字体大小，而不必关心链接的父元素 p 的字体大小。由于根元素（html）设置为 100%（通常为 16px），则将 a 元素设置为 1rem，则表示其字体大小也是 16px。此外，标记了 /* 可选 */ 的样式表示可以提供一个以像素为单位的字体大小，从而为 IE8 及其他不支持 rem 的旧浏览器提供备选方案

2. 指定特定字体大小

(1) 输入 `font-size:`。

(2) 在冒号（:）后输入准确的字号，如 13px（关于像素示例，参见图 10.6.1）。

或者使用关键字指定字体大小，即使用 `xx-small`、`x-small`、`small`、`medium`、`large`、`x-large` 或 `xx-large`。

关于单位的详细信息，参见 7.5 节。

提 示 数字与单位之间不应有任何空格。

提 示 如果以像素为单位设置字体大小，使用 Internet Explorer 的访问者将无法使用浏览器的文本大小选项对文本进行放大或缩小。这是要用 em 或百分比控制字体大小的原因之一。从 IE7 开始，访问者可以对**整个**网页放大或缩小，不过这跟只改变文字大小不同。如今，对于大多数网站来说，编码人员已经无需再担心 IE6 和 IE7 了，因为这两个版本的浏览器用户已经非常少了。不过，截至本书写作之时，在中国还有 25% 的用户使用 IE6，要查看它在世界各地的份额，可以访问 www.ie6countdown.com。

提 示 不同的浏览器对关键字的解释方式可能有所不同。

提 示 磅（pt）用做打印样式表的单位。

提 示 `font-size` 属性是继承的。

3. 根据父元素设置字体大小

(1) 输入 `font-size:`。

(2) 在冒号（:）后输入相对值，如 `1.5em` 或 `150%`（em 和百分数示例参见图 10.6.4）。

或者使用相对关键字，即使用 `larger` 或 `smaller`（这种用法没有百分数常见，而百分

数又没有 em 常见）。

4. **根据根元素设置字体大小**

(1) 输入 `font-size:`。

(2) 在冒号（:）后输入相对值，如 `0.875rem`（rem 示例参见图 10.6.7）。

提示　父元素的字体大小可能是由用户（比较少见）或设计人员设置的，可能是从父元素继承的，也可能来自浏览器的默认设置。上文已经提到，大多数浏览器对 body 元素设置的默认字体大小为 16 像素。

提示　设置了相对字体大小的元素的子元素会继承这个大小，而不是继承相对值。因此，p（参见图 10.6.6）中 a 元素会继承 15 像素的字体大小（参见图 10.6.4），而不是相对值 0.9375em。链接将显示为 15px，除非被其他样式覆盖。

提示　可以将字体大小同其他字体值一起进行设置。参见 10.7 节。

提示　关于在设计网页过程中使用 em 优于使用像素，Chris Coyier 给出了多种原因，参见 http://css-tricks.com/why-ems/。

10.7　设置行高

行高指的是段落的行距，即段落内每行之间的距离，参见图 10.7.1 和图 10.7.2。使用大一些的行高有时候会使主体文本更容易阅读。对于超过一行的标题，使用较小的行高则会让它们看起来更美观。

```
... 省略前面的CSS ...

.intro {
    line-height: 1.45;
}

.intro .subhead {
    font-size: 1.125em;
}

.intro p {
    font-size: 1.0625em;
}

.project p {
    font-size: .9375em; /* 15px/16px */
    line-height: 1.65; /* 15px*1.65 =
    ➝24.75px */
}
```

图 10.7.1　假设 body 元素默认大小为 16 像素，`.project` 类 p 元素的字体大小为 0.9375em，即 15 像素。行高将是 15 像素的 1.65 倍，即大约 24.75 像素。我还设置了 `.intro` 容器的行高，它会被后代元素继承



图 10.7.2　使用 `line-height` 拉大行距可以使其更容易阅读。在这个例子中，底部的主文本之间的行距要比顶部的 `intro` 文本之间的行距明显大

设置行高的步骤

(1) 输入 `line-height:`。

(2) 输入 *n*，这里的 *n* 是一个数字，它与元素的字体大小相乘，得出需要的行高。（这是最为常用的方法，即一个没有单位的数字。）

或者输入 *a*，这里的 *a* 是以 em、像素或磅（仅在打印样式表中使用磅）为单位的值。

或者输入 *p*%，这里的 *p*% 是字体大小的百分数。

提示 如果使用数字设定行高（通常都这么做），那么所有的子元素都会继承这个因子。因此，如果父元素的字体大小是 16 像素（或以 em 等表示的等价大小），行高是 1.5，则该元素的行高就是 24（即 16×1.5）像素。如果子元素的字体大小是 10 像素，则该元素的行高就是 15（即 10×1.5）像素。

提示 如果使用百分数或 em 值，那么只会继承产生的行高（即计算出来的值）。因此，如果父元素的字体大小是 16 像素，行高是 150%，则该元素的行高就是 24 像素。所有的子元素都将继承 24 像素的行高，不管字体大小是多少。

提示 根据下一节将要讲到的，可以将行高跟字体、大小、粗细、字体样式和变体一起进行设置。

10.8 同时设置所有字体值

可以使用 `font` 简写属性同时设置字体样式、粗细、变体、大小、行高和字体系列，参见图 10.8.1 和图 10.8.2。这种方法要比单独声明各个属性高效，因此采用这种方法可以保持样式表简洁。

使用 `font` 简写属性**不要求**指定所有字体属性，但至少应该包含字体大小和字体系列属性。图 10.8.1 中的两个注释说明了不能使用简写属性的情况。

```
body {
    font: 100% Geneva, Tahoma, Verdana,
    → sans-serif;
}

h1,
h2 {
    /* 这些声明无法使用font简记法，除非同时
       声明字体大小 */
    font-family: "Gill Sans", "Gill Sans MT",
    → Calibri, sans-serif;
    font-weight: normal;
}

h1 { font-size: 2.1875em; }

h2 { font-size: 1.75em; }

em,
a:link,
.intro .subhead {
    font-weight: bold;
}

.intro {
    line-height: 1.45;
}

.intro .subhead {
    font-size: 1.125em;
}

.intro p {
    font-size: 1.0625em;
}

.project p {
    /* 这些声明无法使用font简记法，除非同时
       声明字体系列 */
    font-size: .9375em;
    line-height: 1.65;
}
```

图 10.8.1 这个样式表与图 10.7.1 中的样式表是等效的，其效果如图 10.8.2 所示。这里只是将 `body` 的字体属性放在了一条 `font` 声明中。代码中无法将 `h1`、`h2` 或 `.project` 类 `p` 元素的字体属性放入 `font` 中，因为 `font` 简写属性至少应该包括字体系列和字体大小属性。更多示例参见图 10.8.3 和图 10.8.4

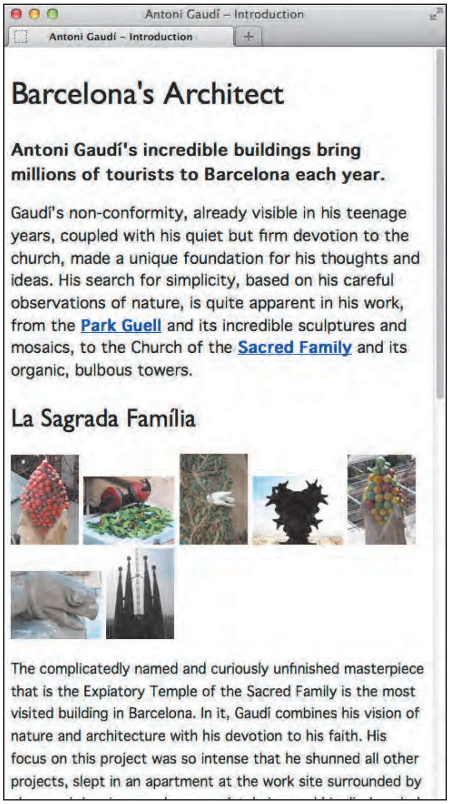


图 10.8.2　这个页面与图 10.7.2 是相同的

```
.example-2 {
    font: .875em/1.3 "Palatino Linotype",
    → Palatino, serif;
}
```

图 10.8.3　这是一个结合了 font-size、line-height 和 font-family 声明的 font 简记法的例子。行高跟在字体大小和一个斜杠后面。font 简写属性中还可以包含 font-style、font-variant 和 font-weight，参见图 10.8.4

```
.example-3 {
    font: italic small-caps bold .875em/1.3
    → "Palatino Linotype", Palatino, serif;
}
```

图 10.8.4　这个例子中包含了所有可能的属性。只要 font 中声明了字体大小和字体系列属性，其他属性可以任意组合。前三个属性的次序无关紧要

同时设置所有字体值的步骤

(1) 输入 font:。

(2) 可选步骤，输入 normal、italic 或 oblique 以设置字型（参见 10.4 节）。

(3) 可选步骤，输入 normal、bold、bolder、lighter 或 100 的倍数（最大到 900）以设置粗细（参见 10.5 节）。

(4) 可选步骤，输入 normal 或 small-caps 来取消或设置小型大写字母（参见 10.15 节）。

(5) 可选步骤，输入需要的字体大小（参见 10.6 节）。

(6) 如果需要，输入 */line-height*，这里的 *line-height* 是行与行之间的距离（参见 10.7 节）。

(7) 输入一个空格，再按优先次序输入需要的字体系列，以逗号分隔（参见 10.2 节）。

提 示　必须始终显式地声明字体大小和字体系列属性：先是字体大小，再是字体系列。

提 示　行高是可选的，但它如果出现，就必须紧跟在字体大小和斜杠后面（参见图 10.8.3 和图 10.8.4）。

提 示　第 (2) ~ (4) 步中的属性可以按任意顺序指定（参见图 10.8.4），也可以忽略（参见图 10.8.1 和图 10.8.3）。如果忽略它们，它们就被自动设为 normal，而这也许不是你想要的。

提 示　font 属性是继承的。

10.9　设置颜色

可以修改网页中文本的颜色。我们的样式表可以包含颜色名称、十六进制值、RGB、HSL、RGBA 和 HSLA 值的任意组合来定义颜色，参见图 10.9.1 和图 10.9.2。关于各种颜色值的信息参见 7.5 节中的“CSS 颜色”。

```
body {
     color: blue;
     font: 100% Geneva, Tahoma, Verdana,
     ➝ sans-serif;
}

...

h2 {
     color: #7d717c;
     font-size: 1.75em;
}

...

/* :::: 链接 :::: */
a:link {
     color: #e10000; /* a red */
}

a:visited {
     color: #b44f4f;
}

a:hover {
     color: #f00;
}

.intro a {
     color: #fdb09d; /* 略带粉色 */
}

.intro a:hover {
     color: #fec4b6;
}
```

图 10.9.1　在 body 元素中设置页面的默认文本颜色，这样所有的元素都会继承设置（link 除外）。再在相应元素中设置特殊需求的颜色（如我在 h2 中的设置）。注意，a:hover 颜色（#f00）使用了第一条提示中说到的缩写（为展示效果，这个例子中的 body 元素使用了蓝色，后面不会保留这一设置）

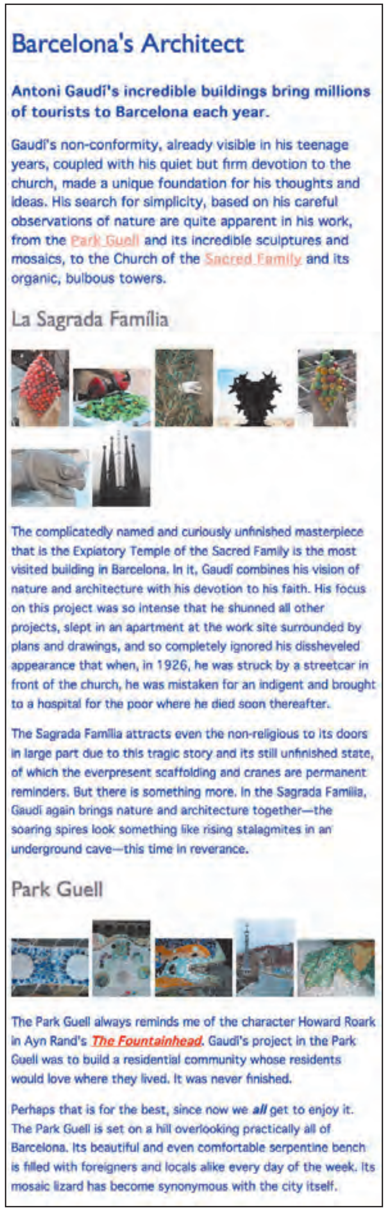


图 10.9.2　不出所料，扎眼的蓝色应用到了页面的大部分文本中，h2 标题采用中度灰色（#7d717c）覆盖了设置。页面顶部 .intro 区域中粉红色（#fdb09d）的链接覆盖了默认的红色（#e10000）链接。下一节应用背景颜色后我会将这里的文本设为白色，不过现在不行，现在设置成白色就看不到了

10

设置颜色的步骤

(1) 输入 `color:`。

(2) 输入 *`colorname`*，这里的 *`colorname`* 是预定义颜色中的一种。

或者输入 *#rrggbb*，这里的 *#rrggbb* 是颜色的十六进制呈现。这是最常用的指定颜色的方法。

或者输入 rgb(*r*, *g*, *b*)，其中 *r*、*g*、*b* 是 0 ~ 255 之间的整数，分别表示所需颜色里红、绿、蓝的量。

或者输入 rgb(*r*%, *g*%, *b*%)，其中 *r*、*g*、*b* 分别是所需颜色里红、绿、蓝的百分数。

或者输入 hsl(*h*, *s*, *l*)，其中 *h* 是 0~360 之间的数值，表示所需颜色的色相；*s* 和 *l* 均是百分数，分别表示所需颜色的饱和度和亮度。（一般来说，对于不透明颜色，使用十六进制数或 RGB 值更好。）

或者输入 rgba(*r*, *g*, *b*, *a*)，其中 *r*、*g*、*b* 是 0 ~ 255 之间的整数，分别表示所需颜色里红、绿、蓝的量；*a* 是 0~1 之间的小数，表示所需颜色的 alpha 透明度。

或者输入 hsla(*h*, *s*, *l*, *a*)，其中 *h* 是 0 ~ 360 之间的数值，表示所需颜色的色相；*s* 和 *l* 均是百分数，分别表示所需颜色的饱和度和亮度；*a* 是 0 ~ 1 之间的小数，表示所需颜色的 alpha 透明度。

如果输入的 *r*、*g* 或 *b* 的值大于 255，就会使用 255。类似地，高于 100% 的百分数将被替换为 100%。

提示　当十六进制数值由三对重复的数字组成时，还可以使用 #rgb 设置颜色。事实上，我推荐使用这种方法。因此，可以（且应该）将 #FF0099 写做 `#F09` 或 `#f09`。类似地，可以将 #CC0000 写成 `#C00` 或 `#c00`。但是不能将 #31AA55 写成 `#31a5`，因为只有两对值是重复的。

提示　记住，Internet Explorer 在 IE9 之前的版本不支持 HSL、RGBA 和 HSLA，因此，如果要在颜色声明中使用这些记法，就需要为 IE 的旧版本定义备用颜色。详细说明参见 7.5 节中的“CSS 颜色”。

提示　图 10.16.2 展示了 `:hover` 链接即将激活的效果。

提示　除了 `link` 元素，其他元素都会继承 `body` 元素中的 `color` 属性。开发人员必须显式设置 `link` 元素的 `color` 属性，如图 10.9.1 所示。

10.10　设置背景

我们在设置背景样式时有很多选择。可以为单个元素设置背景，可以为整个页面设置背景，还可以为上述二者的任意组合设置背景。如此，便可以对几个段落、几个单词、不同状态的链接、内容区域等修改背景。总之，几乎可以对所有元素应用背景样式，甚至是表单和图像。（是的，你没看错，图像也可以有背景图像！）

设置背景有很多属性可以利用，包括 `background-color`、`background-image`、`background-repeat`、`background-attachment` 及 `background-position` 等。还可以使用 `background` 简记法，该属性将上述属性合并了，可以节省大量的输入。本节将介绍这些属性。

我们将首先为本章一直在使用的示例页面添加背景颜色，然后介绍一些额外的与背景相关的例子以深入探讨这一主题。

1. 修改文本的背景颜色

(1) 输入 `background:`。

(2) 输入 transparent（表示允许透过父元素的背景颜色）或 color，这里的 color 是颜色名称、十六进制数值，或 RGB、HSL、RGBA、HSLA 颜色值（参见 10.9 节第 (2) 步），如图 10.10.1 ～图 10.10.6。十六进制值颜色是最为常用的。

```
body {
    background-color: #88b2d2;
    font: 100% Geneva, Tahoma, Verdana,
    → sans-serif;
}

...

h2 {
    background-color: #eaebef;
    color: #7d717c;
    font-size: 1.75em;
}

.intro {
    background-color: #686a63;
    color: #fff;
    line-height: 1.45;
}

...
```

图 10.10.1　除了那些单独设置背景颜色的元素之外，对 body 元素设置背景颜色就是为整个页面设置背景颜色。intro 类 div 元素的背景设置将其与页面的其他部分区别开来，参见图 10.10.2



图 10.10.2　整个页面（body 元素）的背景颜色为浅蓝色，有两处浅蓝色背景被其他颜色覆盖：intro 类 div 的棕色背景，以及二级标题的浅灰色背景

```
.architect {
    background-color: #fff;
}
```

图 10.10.3　包含所有内容的 article 元素有 class="architect"，因此这段 CSS 将让它拥有白色的背景（但一些部分除外），如图 10.10.4 所示



图 10.10.4　白色填充了整个内容区域。.intro 区域和 h2 也是 architect 类 article 的后代，如果之前没有为它们添加不同的背景（如图 10.10.1 所示），它们的背景也应该是白色的。在 .architect article 的外围还能看到浅蓝色的 body 背景，这是由于浏览器默认为 body 设置了一些外边距

10

```
...

.architect {
    background-color: #fff;
    padding: 1.5em 1.75em;
}

.intro {
    background-color: #686a63;
    color: #fff;
    line-height: 1.45;
    padding: 1px 1.875em .7em;
}

...
```

图 10.10.5　这里用到了一个下一章才会讲解到的属性（参见 11.8 节），希望读者们原谅我

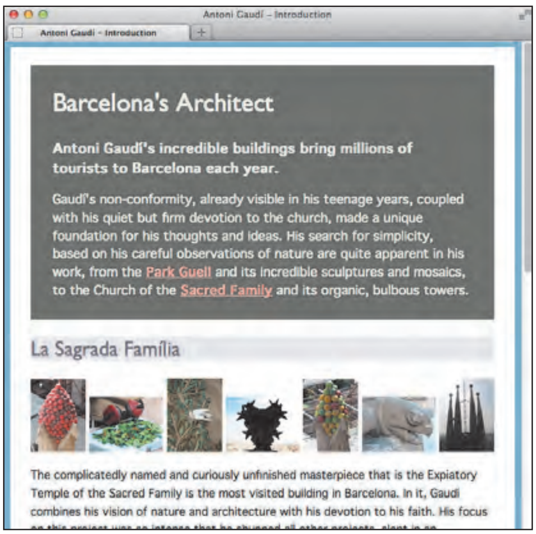


图 10.10.6　添加一点点内边距就能让页面大不一样，现在，页面看起来舒服多了

2. 使用背景图像作为背景

(1) 输入 background-image:。

(2) 输入 url(*image.png*)，这里的 *image.png* 是图像相对于样式表所在位置的路径和文件名，参见图 10.10.7。或者输入 none，表示不使用图像，如 background-image: none;（只有在覆盖应用背景图像的元素的样式时才会使用这个声明）。

```
body {
    background-color: #ccc;
    background-image: url(bg-pattern.png);
    ...
}
```

图 10.10.7　图像 URL 反映的是样式表相对于图像的位置。这里仅作简单处理，更多信息参见第二条提示。这里我希望图像能填充整个页面，但，不必设置 background-repeat，因为背景图像默认就是重复的（如图 10.10.8 所示）。访问者只有在图像加载速度慢（例如在移动连接环境下）或图像由于某些原因没有加载成功时，才会看到背景颜色



图 10.10.8　背景图像（上）包含一个漂亮的图案。将背景图像添加到页面主体后，它会在水平和竖直两个方向上不断重复。我将内容变窄了一些，以便看清效果

3. 重复背景图像

输入 background-repeat: *direction*，这里的 *direction* 可以取四个值：repeat、repeat-x、repeat-y 和 no-repeat，它们分别代

表同时横向和纵向重复图像（参见图 10.10.8）、只横向重复图像（参见图 10.10.9 ～图 10.10.12）、只纵向重复图像、不重复图像。忽略 background-repeat 相当于将其设为默认值 repeat，参见图 10.10.7。

```
body {
    background-image: url(sky.png);
    background-repeat: repeat-x;
    background-position: left bottom;
    ...
}
```

图 10.10.9　这次我们希望图像仅在水平方向（X 轴）重复。重复的开始位置是页面的左上角。注意这里没有提供背景色



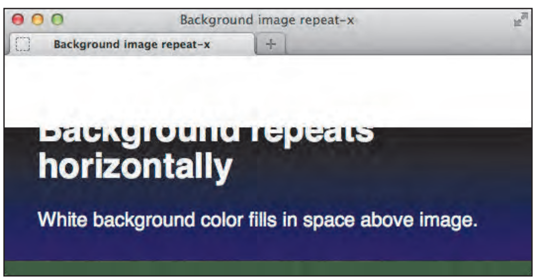


图 10.10.10　这张背景图像（上）由黑色向蓝色再向绿色渐变，很适合水平重复。不过它的高度不足以填充整个页面，在不设置背景色的情况下显得很难看。那一行文字其实并没有被切割，看不见它的上半部分是因为它的颜色与默认的背景色一样，都是白色。另见彩插

```
html {
    min-height: 100%;
}

body {
    background-color: #000;
    ...
}
```

图 10.10.11　在图 10.10.9 的基础上设置背景色，该背景色会在背景图像不覆盖的区域显示出来。这里还添加了一条规则，让 html 元素的最小高度为浏览器窗口高度的 100%，如图 10.10.12 所示（注意：如果将 background-attachment 的值设为 fixed，这一条规则就可以省略，下面很快就会讲到这一点）

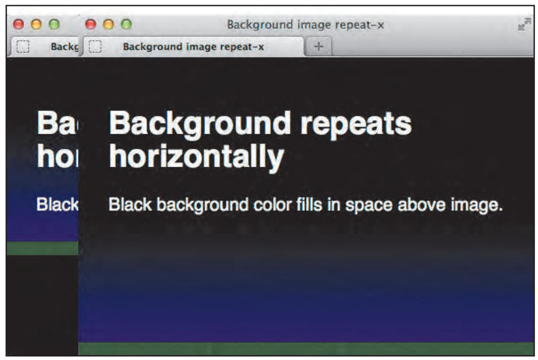


图 10.10.12　黑色（#000）背景色与图像顶端的黑色完美地融合在一起。现在，你可以看到那些文本了！这张图是由两个浏览器窗口截图合成的，这两个窗口的高度是一样的。如果没有为 html 元素添加图 10.10.11 中的规则，效果就如左图所示，背景图像的底部位于文字下面一点的位置（因为页面在那个位置结束了），因此黑色背景色会显示在它的下面。另见彩插

4. 控制背景图像是否随页面滚动

(1) 输入 background-attachment:。

(2) 输入 fixed（代码如图 10.10.13 所示），背景图像会附着在浏览器窗口上（也就是说即使访问者滚动页面，图像仍会继续显示，参见图 10.10.14）；或 scroll，访问者滚动页面时背景图像会移动，参见图 10.10.15；

10

或 local，只有访问者滚动背景图像所在的元素（而不是整个页面）时，背景图像才移动。忽略此项，浏览器通常会使用默认值 scroll）。

```
body {
    background-color: #000;
    background-image: url(sky.png);
    background-repeat: repeat-x;
    background-attachment: fixed;
    background-position: left bottom;
    …
}
```

图 10.10.13　配合我们设置的 background-position，背景图像将从浏览器窗口的左下角开始水平重复

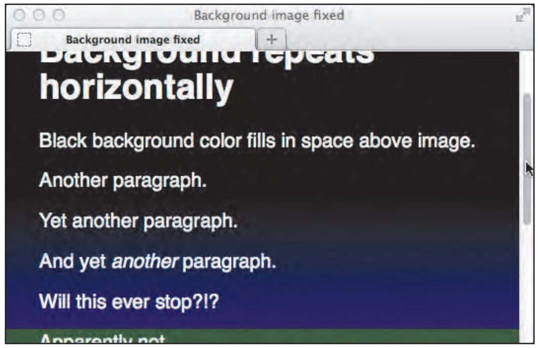
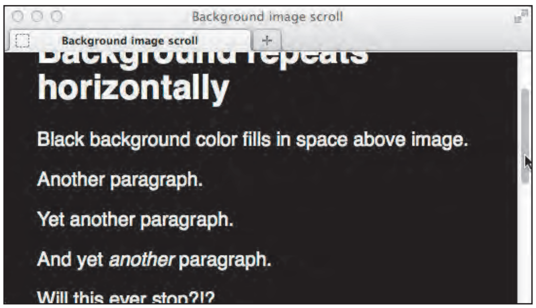


图 10.10.14　无论页面中有多少内容，无论用户如何滚动页面，背景图像固定不动



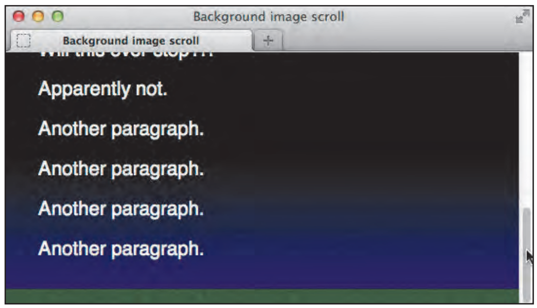


图 10.10.15　这是图 10.10.14 所示的页面在不设置 background-attachment: fixed; 情况下的显示效果。（不设置该规则等价于设置 background-attachment: scroll;，因为 scroll 是默认值。）背景图像的位置根据内容的长度而变，因此，直到滚动到页面的最底部，才能看到背景图像（如右图所示）

5. 指定元素背景图像的位置

输入 background-position_x y_，其中 *x* 和 *y* 可以表示为距离左上角的绝对距离或百分数，如图 10.10.16 所示，如 20px 147px（允许负值）。也可以用 left（左对齐）、center（居中）或 right（右对齐）表示 *x*，用 top（顶端对齐）、center（居中）或 bottom（底端对齐）表示 *y*，如图 10.10.13 所示。如果两个值都使用关键字表示，那么两者的顺序无关紧要，如 top right 跟 right top 一样。

```
body {
    background: #004 url(../img/ufo.png)
    → no-repeat 170px 20px;
    color: greenyellow;
    …
}
```

图 10.10.16　现在，我们使用 background 简记法将所有这些单独的声明压缩成一条规则。不使用 background-position，而是将水平和竖直属性值直接写在最后。注意，使用 no-repeat 会让图像仅显示一次，不重复显示（如图 10.10.17 所示）

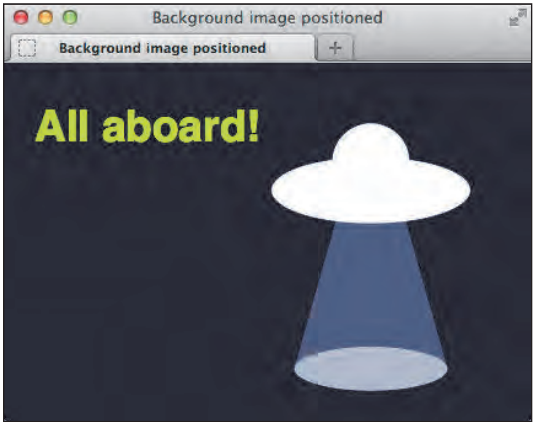


图 10.10.17 UFO 出现在距离页面左边缘 170 像素，距离上边缘 20 像素的位置，从而为绿色文本腾出空间。图像的空白区域都是透明的（光束则具有 alpha 透明度），因此它可以和深蓝色的 body 背景色融合在一起

6. 在一条声明中设置所有背景属性

(1) 输入 `background:`。

(2) 指定任何可接受的 background 属性值（从“修改文本的背景颜色”到“指定元素背景图像的位置”介绍的内容都适用）。这些值的排列顺序没有要求（如图 10.10.16 所示）。

提示 尽管大部分例子都是对 body 样式所做的修改，但实际上你可以为几乎所有的 HTML 元素添加背景。

提示 为了让例子变得简单，背景图像 URL 均使用基本的 URL。实际上，你的 URL 可能是类似于 ../img/ufo.png（如图 10.10.16 所示）的形式，因为通常会避免将图像与样式表放在同一个目录里。本书配套网站上针对本章的例子均使用如图 10.10.18 所示的文件夹结构，因此 CSS 中的背景图像 URL 也会做相应调整。

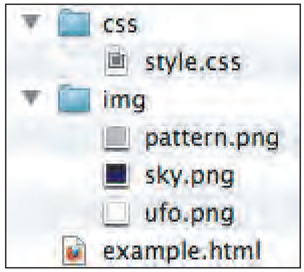


图 10.10.18 在这样的结构中，style.css 中显示背景图像的声明可能形如 `background: url(../img/pattern.png);`。更多信息参见 1.7 节

提示 background-color 的默认值是 transparent（透明），background-image 的默认值是 none（无），background-repeat 的默认值是 repeat（重复），background-attachment 的默认值是 scroll（滚动），background-position 的默认值是 0 0（等价于 top left，即左上角）。只有在需要覆盖其他样式规则的时候才会显式地指定默认值。

提示 通常，尽量使用 background 简记法。使用该简记法的时候，不必指定所有的属性。例如，在实践中，图 10.10.7 中的规则通常写作 `background: #ccc url(bg-pattern.png);`。不过需要注意的是，在简记法中，任何没有指定的属性都将设置为默认值，这有可能会覆盖先前定义的样式。

提示 默认情况下，元素的背景会填充其内容和内边距所在的区域（如图 10.10.5 和图 10.10.6 所示），并延伸到边框的边缘（参见 11.5 节）。可以通过 background-clip 改变这种规则（参见补充材料“更多 CSS3 的背景控制”）。

10

提示 在 CSS3 中，background-attachment 还可以设置成 local。Lea Verou 的一篇文章演示了该属性值的效果（参见 http://lea.verou.me/2012/04/background-attachment-local/）。支持该属性值的浏览器仅包括 IE9+、Chrome、Safari 和 Opera（没有 Firefox）[①]。

提示 可以在 background-position 属性中使用负数。例如，background-position: -45px 80px 会将图像定位在元素左边外侧 45 像素（因此不会看到图像在水平方向上的头 45 个像素）、顶部内侧 80 像素。

提示 文本和背景之间应有足够的对比度。这样做不仅能帮助普通用户，对视力障碍的访问者尤其重要（参见 http://contrastrebellion.com）。这里我们再次提到 Lea Verou，他创建了一个帮助选择颜色的工具，使用该工具可以选择对比度符合无障碍访问指南要求的颜色。

提示 如果图 10.10.14 中不使用 background-position: left bottom;，那么效果是相反的。我们将会看到图像在页面的**顶端**水平重复（顶端是默认位置）。当页面向下滚动时，该背景图像就会随着内容一起离开我们的视野。

更多 CSS3 的背景控制

CSS3 提供了更多的背景效果。本书将在第 14 章介绍渐变背景和多重背景。

background-clip 和 background-origin 两个属性分别控制元素背景显示的范围和开始的位置。这两个属性都接受相同的值：content-box 包含内容，padding-box 包含内容和内边距，border-box 包含内容、内边距和边框（参见 11.5 节）。background-clip 的默认值为 border-box，而 background-origin 的默认值则为 padding-box。更多信息参见 http://css-tricks.com/transparent-borders-with-background-clip/。

background-size 属性可以通过以下属性值控制背景图像的显示尺寸。

- contain：在显示图像完整宽度和高度的情况下，尽可能地扩大图像的尺寸。使用该值，背景图像可能不会填充整个背景区域。
- cover：在填充元素整个背景区域的情况下，让图像尽可能地小。使用该值，图像的一部分可能会超出元素的范围，因而不可见。
- 一个长度、百分数或 auto：如 background-size: 250px 400px; 或 background-size: 50% 50%;。

关于该属性的更多信息参见 www.css3.info/background-size/。该属性擅长巧用 sprite（参见 12.5 节）。

大多数关于新的背景样式的讨论都可以在 www.sitepoint.com/new-properties-and-values-inbackgrounds-with-css3/ 找到。

① 2013 年 10 月发布的 Firefox 25 版已支持该属性值。——译者注

10.11 控制间距

可以增加或减少单词之间或字母之间的距离，前者称为字间距（tracking），后者称为字偶距（kerning），参见图 10.11.1。

```
body {
      background-color: #88b2d2;
      font: 100% Geneva, Tahoma, Verdana,
      → sans-serif;
}

h1,
h2 {
      font-family: "Gill Sans", "Gill Sans MT",
      → Calibri, sans-serif;
      font-weight: normal;

      /* 出于演示目的进行临时设置 */
      letter-spacing: 7px;
}

h1 {
      font-size: 2.1875em;
}

... 其他CSS ...
```

图 10.11.1 这里为标题的字母之间添加了 7 像素的额外间距，letter-spacing 的效果非常明显了。在后面的例子中我会将这个值缩小为 1 像素

1. 指定字间距

输入 word-spacing: *length*，这里的 *length* 是一个带单位的数字，如 0.4em 或 5px。

2. 指定字偶距

输入 letter-spacing: *length*，这里的 *length* 是一个带单位的数字，如 0.4em 或 5px。

提示 可以对单词间距和字母间距使用负数。

提示 单词间距和字母间距的值还可能受到所选的对齐方式（用 text-alignment 设置）和字体系列的影响。



图 10.11.2 标题中字母之间的距离变大了。将这里的字母间距跟图 10.10.6 图中的字母间距进行比较，很容易看出效果。不过，后续例子不会保留这里的设置

提示 要将字母间距和单词间距设为默认值（即不添加额外的间距），可以使用 normal 或 0。

提示 如果要使用 em 值，那么只有结果大小（即“计算出来的值”）会被继承。因此，如果父元素字体大小为 16 像素，额外的单词间距为 0.1em，则每个单词之间的额外间距为 1.6 像素。同时，所有子元素每个单词之间也有 1.6 像素的额外间距，不管它们的字体大小是多少。如果要覆盖继承的值，可以显式地为子元素设置间距。

提示 word-spacing 和 letter-spacing 属性是继承的。

10.12 添加缩进

通过设置 text-indent 属性，可以指定段落第一行前面应该空出多大的空间，如图 10.12.1 和图 10.12.2 所示。

```
...

h1,
h2 {
    font-family: "Gill Sans", "Gill Sans MT",
    → Calibri, sans-serif;
    font-weight: normal;
    letter-spacing: 1px;
}

...

.project p {
    font-size: .9375em; /* 15px/16px */
    line-height: 1.65;
    text-indent: 2em; /* 30px */
}

... 其他CSS ...
```

图 10.12.1　这段代码为 p 元素添加了 2em 的缩进。由于 0.9375 的字体大小等于 15 像素（15/16=0.9375），因此这个缩进约为 30 像素（参见图 10.12.2）

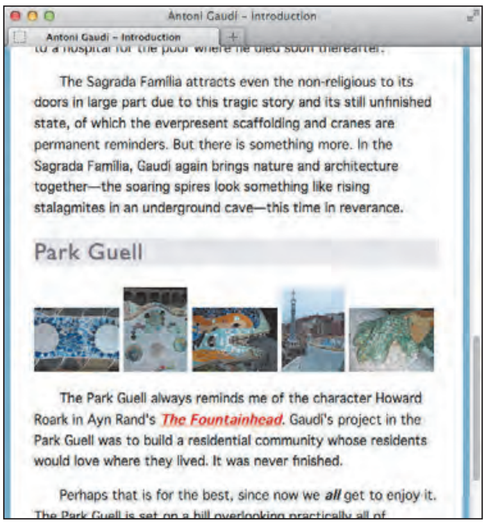


图 10.12.2　每个段落都有 30 像素的缩进

增加缩进的方法

输入 `text-indent:` *`length`*，这里的 *`length`* 是一个带单位的数字，如 `1.5em` 或 `18px`。

提示 我们可以为其他元素应用 `text-indent` 属性，而不仅是段落元素。但默认情况下对 `em`、`strong`、`cite` 等内联元素没有效果。可以将它们设置为 `display: block;` 或 `display: inline-block;`，就可以强制为其应用 `text-indent` 属性。

提示 使用负数会产生悬挂缩进。使用悬挂缩进时，可能还需要增加文字框周围的内边距或外边距，从而让容器可容纳伸到外边的文本（参见 11.8 节和 11.10 节）。

提示 与通常情况一样，em 值要根据元素的字体大小进行计算（参见图 10.12.1），百分数要根据父元素的宽度进行计算。

提示 `text-indent` 属性是继承的。

提示 如果要使用百分数或 em 值，那么只有生成的大小（即“计算出来的值”）会被继承。因此，如果父元素为 300 像素宽，那么 10% 的 `text-indent` 就是 30 像素，同时，所有子元素第一行也都会缩进 30 像素，而不管它们各自父元素的宽度是多少。

提示 要忽略继承的缩进，使用 0 即可。

10.13 对齐文本

根据需要，可以让文本左对齐、右对齐、居中对齐或两端对齐，参见图 10.13.1。

对齐文本的步骤

(1) 输入 `text-align:`；

(2) 输入 `left` 让文本左对齐；

或者输入 `right` 让文本右对齐；

或者输入 `center` 让文本居于屏幕的中间；

或者输入 justify 让文本两端对齐。

```
...

h1, h2 {
     font-family: "Gill Sans", "Gill Sans MT",
     → Calibri, sans-serif;
     font-weight: normal;
     letter-spacing: 1px;
     text-align: center;
}

...

p {
     text-align: justify;
}

.intro .subhead {
     font-size: 1.125em;
     text-align: center;
}

... 其他CSS ...
```

图 10.13.1　有的文本会居中对齐，但大多数文本都会两端对齐，参见图 10.13.2

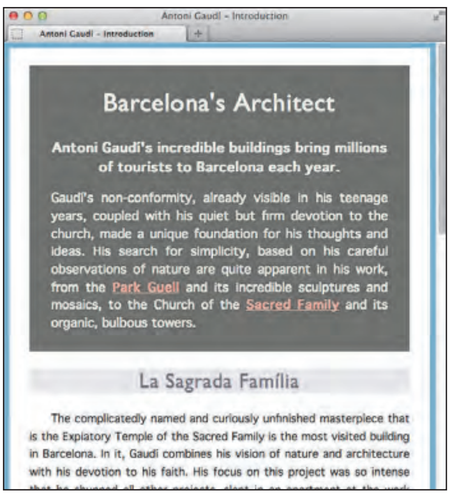


图 10.13.2　做出更改以后，标题和 .subhead 段落变成居中对齐，其他段落文本为两端对齐。之前为段落设置的 text-indent 值保留下来了。注意：我删除了第二个标题下面的图像，以便你能从这个图中看到更多文本

提示　如果选择让文本两端对齐，要注意单词间距和字母间距有可能受到严重的影响。更多信息参见 10.11 节。

提示　text-align 对适用于盒模型的元素（如 h1 ～ h6、p 等，显示为单独的行）来说是起作用的，但对短语内容元素（如 strong、em、a、cite 等，显示在行内）来说则不起作用，除非将它们的显示改为块级。（短语内容元素在 HTML5 之前称为“行内”元素，参见 1.9 节。）为了让短语内容元素中的文字对齐，区别于周边的文本，必须先将它们默认的 display: inline; 样式改为 display: block; 或 display: inline-block;（前者会让它们像段落一样显示为单独的行），然后再设置相应的 text-align 值。对于设置了 display: inline-block; 的元素，需要设置一定的宽度才能看到对齐效果。实际上，需要为“行内”元素设置 text-align 属性的情况是非常少见的。

提示　text-align 属性是继承的。它的默认值取决于文档的语言和书写系统（从右到左或从左到右），不过在大多数情况下，它会被不加区分地设置为左对齐。

10.14　修改文本的大小写

使用 text-transform 属性，可以为样式定义文本的大小写（参见图 10.14.1）。通过这种方法，可以将文本显示为首字母大写、全部大写（参见图 10.14.2）、全部小写或按原样显示。

10

```
body {
    background-color: #88b2d2;
    font: 100% Geneva, Tahoma, Verdana,
    → sans-serif;
}

h1,
h2 {
    font-family: "Gill Sans", "Gill Sans MT",
    → Calibri, sans-serif;
    font-weight: normal;
    letter-spacing: 1px;
    text-align: center;
}

h1 {
    font-size: 2.1875em;
    text-transform: uppercase;
}

... 其余的CSS ...
```

图 10.14.1　将一级标题显示为全部大写，以便突出它，代码实现参见图 10.14.2



图 10.14.2　现在，一级标题真的很醒目，二级标题没有改变

修改文本大小写的步骤

(1) 输入 text-transform:。

(2) 在冒号（:）后输入 capitalize 让每个单词的首字母大写；

或者输入 uppercase 让所有字母大写；

或者输入 lowercase 让所有字母小写；

或者输入 none 让文本保持本来的样子（可以用来取消继承的值）。

提示　capitalize 属性值有它的局限性。它并不了解一门语言里按照惯例首字母不应该大写的单词，它只是简单地让每个单词首字母大写。因此，HTML 中的文本 Jim Rice enters the Hall of Fame 将显示为 Jim Rice Enters The Hall Of Fame。

提示　既然可以改变 HTML 里的文本，为什么还要用 text-transform 呢？这是因为，有时，内容是你无法控制的。例如，内容可能存储在数据库里，或者来自另一个网站的新闻源。在这些情况下，只能通过 CSS 控制文本的大小写。搜索引擎通常是按它在 HTML 里输入的样子索引文本的，在搜索结果里显示标准的大小写会更容易阅读。

提示　lowercase 属性值可以用来创建特殊的标题样式（想象自己是诗人 e.e. cummings[①]）。

提示　text-transform 属性是继承的。

10.15　使用小型大写字母

很多字体都有对应的小型大写字母变体，其中一些字母是大写的，但缩小到了小写字母的大小。可以使用 font-variant 调用小型大写字母变体（参见图 10.15.1）。

① 美国诗人 E. E. Cummings 的一些作品全部使用小写字母署名，即 e.e. cummings。——译者注

```
...

h1,
h2 {
    font-family: "Gill Sans", "Gill Sans MT",
    ➝ Calibri, sans-serif;
    font-weight: normal;
    letter-spacing: 1px;
    text-align: center;
}

h1 {
    font-size: 2.1875em;
    text-transform: uppercase;
}

h2 {
    background-color: #eaebef;
    color: #7d717c;
    font-size: 1.75em;
    font-variant: small-caps;
}

... 其余的CSS ...
```

10.15.1　这里为 h2 设置了 small-caps，如图 10.15.2 所示。不要忘记 font-variant 和 small-caps 中的连字符（-）



图 10.15.2　现在可以看到 h2 中每个字母的小型大写版本。小型大写字母在不同浏览器中的显示效果稍有不同

1. 使用小型大写字母的方法

输入 font-variant: small-caps。

2. 取消小型大写字母的方法

输入 font-variant: none。

提示　与简单地缩小字号的大写字母相比，小型大写字母显得更为轻巧。

提示　并非所有的字体都有对应的小型大写字母设计。如果浏览器没有找到这种设计，那么它有几种选择：可以简单地缩小大写字母的尺寸来模拟小型大写字母（这会使它们显得有点矮胖），也可以完全忽略小型大写字母，并将它们显示为全部大写（就像前面描述的 text-transform: uppercase 那样）。

提示　font-variant 属性是继承的。

10.16　装饰文本

可以使用 text-decoration 属性装饰文本，如添加下划线或者其他类型的线条。目前，最常用到这个属性的情况是为链接状态添加样式，参见图 10.16.1 和图 10.16.2。

1. 装饰文本的步骤

(1) 输入 text-decoration:。

(2) 在冒号（:）后输入 underline 以添加下划线；

或者输入 overline 以添加上划线；

或者输入 line-through 以添加删除线。

2. 取消文本装饰的方法

输入 text-decoration: none;。

提示　除了 a 元素外，也可以为其他元素应用 text-decoration 属性。

提示　对于正常情况下有装饰的元素（如 a、del、ins）以及从父元素继承了装饰样式的元素，可以取消它们的装饰。

10

```
body {
     background-color: #88b2d2;
     font: 100% Geneva, Tahoma, Verdana,
     → sans-serif;
}

h1, h2 {
     font-family: "Gill Sans", "Gill Sans MT",
     → Calibri, sans-serif;
     font-weight: normal;
     letter-spacing: 1px;
     text-align: center;
}

h1 {
     font-size: 2.1875em; /* 35px/16px */
     text-transform: uppercase;
}

h2 {
     background-color: #eaebef;
     color: #7d717c;
     font-size: 1.75em; /* 28px/16px */
     font-variant: small-caps;
}

p { text-align: justify; }

em,
a:link,
.intro .subhead {
     font-weight: bold;
}

.architect {
     background-color: #fff;
     padding: 1.5em 1.75em;
}
```

（接左栏）

```
.intro {
     background-color: #686a63;
     color: #fff;
     line-height: 1.45;
     padding: 1px 1.875em .7em;
}

.intro .subhead {
     font-size: 1.125em; /* 18px/16px */
     text-align: center;
}

.intro p {
     font-size: 1.0625em; /* 17px/16px */
}

.project p {
     text-indent: 2em;
     font-size: .9375em; /* 15px/16px */
     line-height: 1.65;
}

a:link {
     color: #e10000;
     text-decoration: none;
}

a:visited { color: #b44f4f; }

a:hover {
     color: #f00;
     text-decoration: underline;
}

.intro a { color: #fdb09d; }
.intro a:hover { color: #fec4b6; }
```

图 10.16.1　这是本章所使用的完整的样式表，包括改变链接样式的 `text-decoration`，参见图 10.16.2。配套网站上的代码版本包含了更多注释

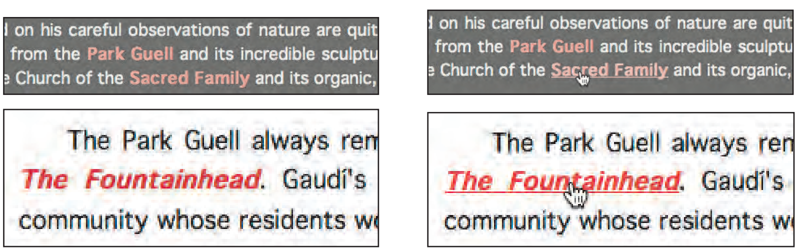


图 10.16.2　在最上面的图中，默认状态下所有链接都没有下划线。鼠标悬停在链接上时又会出现下划线，且链接文本的颜色会更亮一些，它们提醒访问者可以进一步采取行动

提示　将链接的下划线去掉固然很好，但是必须用别的方式将链接与周围的内容进行区分（参见图 10.16.1 和图 10.16.2），否则访问者就无法知道它们是可激活的链接。

10.17　设置空白属性

默认情况下，HTML 文档里的多个空格和回车会显示为一个空格，或者被忽略。如果要让浏览器显示这些额外的空格，可以使用 white-space 属性，参见图 10.17.1 和图 10.17.2。

```
...

.intro {
    background-color: #686a63;
    color: #fff;
    line-height: 1.45;
    padding: 1px 1.875em .7em;
}

.intro .subhead {
    font-size: 1.125em;

    /* 出于演示目的进行的临时设置 */
    color: lime;
    text-shadow: 3px 2px 2px black;
    white-space: nowrap;
}

... 其余的CSS ...
```

图 10.17.1　这里阻止了 .subhead 段落文本断行，是为了突出 nowrap 在深色和浅色背景下的影响，参见图 10.17.2。14.5 节会讲解 text-shadow 属性

设置空白属性的步骤

(1) 输入 white-space:；

(2) 输入 pre，以让浏览器显示原文本中所有的空格和回车；

或者输入 nowrap，确保所有空格不断行，也就是文本全部显示在一行；

或者输入 normal，按正常方式处理空格。

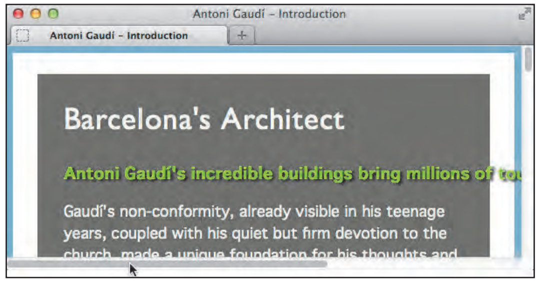


图 10.17.2　.subhead 段落文本不会断行，即使浏览器窗口太窄不足以显示整行文本。这样，页面会添加一个水平滚动条，以便用户能通过滚动查看其余文本

提示　white-space 属性的 pre 值的名称来源于 pre 元素（参见第 4 章），该元素以等宽字体显示其包含的文本，并保留所有的空格和回车。不过，white-space: pre; 不会用等宽字体显示文本。

提示　如果为图 10.17.1 中的 .intro.subhead 规则添加 overflow: hidden; text-overflow: ellipses;，超出元素盒的文本会显示为省略号。

提示　在具有 white-space: nowrap 样式的元素中，可以使用 br 元素手动创建换行。尽管如此，还是要尽量避免使用 br，除非没有更好的选择，因为这样做会在 HTML 中混合表现，而不是让 CSS 控制它。关于 br 元素的更多信息参见 4.16 节。

10

第 11 章

用 CSS 进行布局

本章内容

- ❑ 开始布局的注意事项
- ❑ 建立页面结构
- ❑ 在旧版浏览器上为 HTML5 元素添加样式
- ❑ 对默认样式进行重置或标准化
- ❑ 盒模型
- ❑ 控制显示类型和元素的可见性
- ❑ 设置元素的高度和宽度
- ❑ 添加元素周围的内边距
- ❑ 设置元素的边框
- ❑ 设置元素周围的外边距
- ❑ 使元素浮动
- ❑ 控制元素浮动的位置
- ❑ 对元素进行相对定位
- ❑ 对元素进行绝对定位
- ❑ 在栈中定位元素
- ❑ 处理溢出
- ❑ 垂直对齐元素
- ❑ 修改鼠标指针

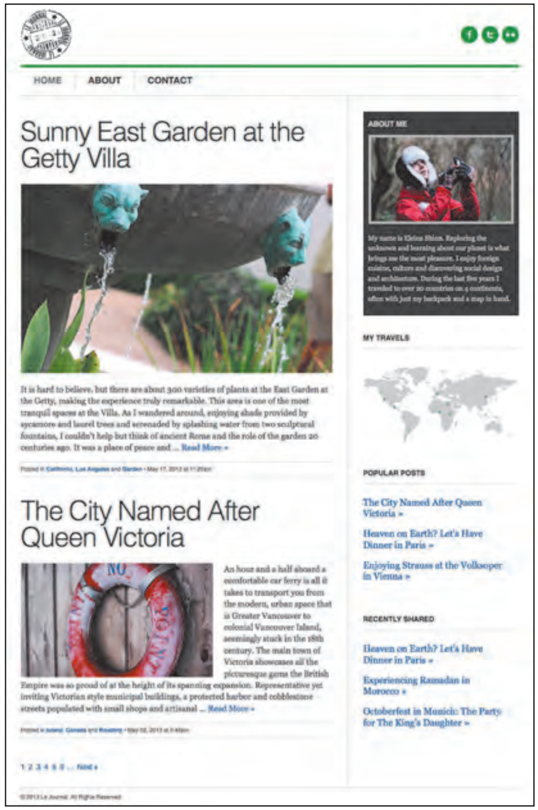


图 11.0.1　这个页面包含两栏、一个页眉和一个页脚，它是用 CSS 进行布局的。CSS 具有强大的改变页面外观的功能。这个页面的设计相当简单，以内容为重点。本章一步步地介绍了如何构建该布局，我们还会在第 12 章将这个页面做成响应式 Web 页面

可以使用 CSS 创建各种各样的布局。本章将演示如何创建一种常见的布局：顶部有一个报头，中间是两栏内容，底部有一个页脚，如图 11.0.1 所示。不过，使用即将讲到的 CSS 属性，还可以创建截然不同的布局。

本章和下一章是相互关联的。为了缓解学习曲线，本章演示了创建固定宽度布局的方法，不过你学到的大部分知识也适用于其他布局。在第 12 章中，我们会学习如何将这个页面做成响应式 Web 页面。

在本章中，不会将 CSS 的每一行都显示出来。例如，大部分文本格式化都在前面完成了。完整的代码见 www.htmlcssvqs.com/8ed/11/finished-page。所有这些文件（尤其是样式表）中都包含了大量的注释，用以对代码进行解释。

11.1　开始布局的注意事项

下面列举的一些要点有助于对网站进行布局，并在发布之前对网站进行调整。

1. 内容与显示分离

之前的章节已经提到过了，不过，由于这是构建网页的根本，因此有必要再次强调一下。作为最佳实践，应始终保持内容（HTML）与显示（CSS）分离。第 8 章介绍了如何通过外部样式表实现这一点。如果对所有的页面都这样做，就可以共享相同的布局和整体样式，而只在具体页面设置差异即可。

2. 布局方法

网站设计主要有两大类型：固定宽度和响应式。

- 对于**固定**（fixed）布局，整个页面和每一栏都有基于像素的宽度。顾名思义，无论是使用移动电话和平板电脑等较小的设备查看页面，还是使用桌面浏览器并对窗口进行缩小，它的宽度都不会改变。要知道，你访问过的大部分网站都使用固定布局。在引入响应式 Web 设计之前，这是大多数网站选用的布局方式。尽管使用固定布局的网站的比例在减少，但这种布局仍很常见，尤其是公司网站和大牌网站。此外，固定布局也是学习 CSS 时最容易掌握的布局方式，这也是本章演示这种布局的原因。
- **响应式**页面也称为**流式**（fluid 或 liquid）页面，它使用百分数定义宽度，允许页面随显示环境的改变进行放大或缩小。除了具有流动栏，响应式页面还可以根据屏幕尺寸以特定方式调整其设计。例如，可以更改图像大小或者调整每一栏，使其大小更合适。这就可以在使用相同 HTML 的情况下，为移动用户、平板电脑用户和桌面用户定制单独的体验，而不是提供三个独立的网站。

没有一种布局方式可以适用于所有的情景。不过，随着智能手机和平板电脑的广泛出现（更别提我们目前还不知道，但未来一定会出现的各种不同尺寸的设备），有必要将网站做成响应式布局。这也是每天都有大量响应式网站出现的原因。上文已经提到，我们将在下一章学习如何创建响应式网站。

3. 浏览器注意事项

并非所有的访问者都使用同样的浏览器，同样的操作系统，甚至同样的设备访问你的网站。因此，大多数情况下，在将网站放到服务器上发布之前，通常需要在很多浏览器上对页面进行测试。推荐在开发过程中就用几个浏览器对页面定期进行测试，这样，在最后进行全面测试时，碰到的问题就会少一些。关于如何对页面进行测试，以及测试用浏览器的有关信息，参见 20.2 节。

> **响应式 Web 设计的起源**
>
> Ethan Marcotte 创建了术语“响应式 Web 设计”（responsive Web design），并向大家介绍了创建响应式网站的技术。人们首次广泛关注这种方法始于他发表在 A List Apart 上的文章（www.alistapart.com/articles/responsive-web-design/）。他在 *Responsive Web Design* 一书中对此作了更为深入细致的探讨，强烈推荐读者去读一读。
>
> 本书第 12 章讲解了 Ethan Marcotte 普及的这种技术，以及 Web 社区其他贡献者对响应式 Web 设计的改进和补充。

11.2　构建页面

CSS 让页面内容富有活力，开发人员或者设计人员的设计技巧大部分都要靠它来体现。可以将 CSS 应用于第 3 章讲到的表现页面主体结构元素的内容容器，以及其中的内容（第 4 章、第 5 章、第 15 ~ 18 章）。但首先我们应该知道，高效网页的核心是结构良好、语义化的 HTML，参见图 11.2.1。

构建页面

(1) 恰当地使用 `article`、`aside`、`nav`、`section`、`header`、`footer` 和 `div` 等元素将页面划分成不同的逻辑区块。根据需要，对它们应用 ARIA 地标角色。上述两点的详细说明见第 3 章。图 11.2.1 创建了一个名为 *Le Journal* 的虚构博客，它的页面结构如下所示。

- ❑ 三个 `div`，其中一个将整个页面包起来，另外两个将两部分主体内容区域包起来以便应用样式设计。
- ❑ 用作报头的 `header`，包括标识、社交媒体网站链接和主导航。
- ❑ 划分为多个博客条目 `section` 元素的 `main` 元素，其中每个 `section` 元素都有自己的页脚。
- ❑ 附注栏 `div`（同时使用了 `article` 和 `aside`），提供关于博客作者和右栏（应用 CSS 之后就有了）博客条目的链接。
- ❑ 页面级 `footer` 元素，包含版权信息等内容。

(2) 按照一定的顺序放置内容，确保页面在不使用 CSS 的情况下也是合理的（参见图 11.2.2）。例如，首先是报头，接着是主体内容，接着是一个或多个附注栏，最后是页面级的页脚。将最重要的内容放在最上面，对于智能手机和平板电脑等小屏幕用户来说，不用滚动太远就能获取主体内容。此外，搜索引擎“看到”的页面也类似于未应用 CSS 的页面，因此，如果将主体内容提前，搜索引擎就能更好地对网站进行索引。这同样也会让屏幕阅读器用户受益。

(3) 以一致的方式使用标题元素（`h1` ~ `h6`），从而明确地标识页面上这些区块的信息，并对它们按优先级排序。

(4) 使用合适的语义标记剩余的内容，如段落、图和列表。

(5) 如果有必要，使用注释来标识页面上不同的区域及其内容，如图 11.2.1 所示。从代码中可以看出，笔者喜欢使用一种不同的注释格式标记区块的开始（而非结束）。

> **提示**　不一定要在应用 CSS 之前就标记好整个页面。实践中，很少先将一个区块的 HTML 写好，再为其编写一些或全部的 CSS，然后再对下一个区块重复这一过程，等等。处理方式取决于个人习惯。

```
...
<body>
<div class="page">
    <!-- ==== 开始报头 ==== -->
    <header class="masthead"
    ➝ role="banner">
       <p class="logo"><a href="/"><img ...
       ➝ /></a></p>

       <ul class="social-sites">
          ... [社交图片链接] ...
       </ul>

       <nav role="navigation">
          ... [主导航链接列表] ...
       </nav>
    </header>
    <!-- 结束报头 -->

    <div class="container">
       <!-- ==== 开始主体内容 ==== -->
       <main role="main">
          <section class="post">
             <h1>Sunny East Garden at the
             ➝ Getty Villa</h1>

             <img ... class="post-photo-full" />

             <div class="post-blurb">
                <p>It is hard to believe ...</p>
             </div>

             <footer class="footer">
                ... [博客条目页脚] ...
             </footer>
          </section>

          <section class="post">
             <h1>The City Named After Queen
             ➝ Victoria</h1>

             <img ... class="post-photo" />

             <div class="post-blurb">
                <p>An hour and a half aboard
                ➝ ...</p>
             </div>

             <footer class="footer">
                ... [博客条目页脚] ...
             </footer>
          </section>
          <nav role="navigation">
             <ol class="pagination">
                ... [链接列表项] ...
             </ol>
          </nav>
       </main>
       <!-- 结束主体内容 -->

       <!-- ==== 开始附注栏 ==== -->
       <div class="sidebar">
          <article class="about">
             <h2>About Me</h2>
             ...
          </article>

          <div class="mod">
             <h2>My Travels</h2>
             ... [映射图像] ...
          </div>

          <aside class="mod">
             <h2>Popular Posts</h2>
             <ul class="links">
                ... [链接列表项] ...
             </ul>
          </aside>

          <aside class="mod">
             <h2>Recently Shared</h2>
             <ul class="links">
                ... [链接列表项] ...
             </ul>
          </aside>
       </div>
       <!-- 结束附注栏 -->
    </div>
    <!-- 结束容器 -->

    <!-- ==== 开始页脚 ==== -->
    <footer role="contentinfo" class=
    ➝ "footer">
       <p class="legal"><small>&copy; 2013 Le
       ➝ Journal ...</small></p>
    </footer>
    <!-- 结束页脚 -->
</div>
<!-- 结束页面 -->
</body>
</html>
```

图 11.2.1 这是贯穿本章和下一章的 HTML 页面，这个页面中有四个主要区块（masthead、main、sidebar 和页面 footer）。主体内容和附注栏区域包含在 class="container" 的 div 中。整个页面包含在 class="page" 的 div 中。完整的页面内容请参见 www.htmlcssvqs.com/8ed/11/finished-page，默认情况下页面很普通，但功能完整，参见图 11.2.2



图 11.2.2　这是示例在浏览器中显示的样子，除了浏览器默认样式以外并没有设置其他样式。这个页面是一栏的。由于它的语义结构非常好，页面是完全可用且可理解的，只是有一点朴素

关于语义的注意事项

你可能已经注意到，我在图 11.2.1 中使用了 section 元素来标记每个包含部分博文的条目（图 11.2.2 展示了一个）。如果它们是完整的博文条目，我就会使用 article 标记它们，就像单独的、完整的博客文章页面那样。对它们使用 article 替代 section 也没有错，只是代表这一段代码足以成为独立的内容。第 3 章包含了各种使用 article 和 section 的例子。

11.3　在旧版浏览器中为 HTML5 元素添加样式

我们已经知道，HTML5 引入了一些新的语义化元素，其中大多数在第 3 章和第 4 章作了讲解。在大多数情况下，现代浏览器原生支持这些元素。从样式的角度来说，这意味着浏览器将为这些新的元素应用默认样式，就像它们对待早在这门语言诞生之际就存在的元素那样。例如，article、footer、header、nav 以及其他一些元素显示为单独的行，就像 div、blockquote、p 以及其他的在 HTML5 之前的 HTML 版本中称做块级元素的元素。

你可能会想：“对于旧版浏览器呢？这些浏览器出现时还不存在 HTML5 的新元素，我该如何使用这些元素呢？”

值得庆幸的是大多数浏览器允许对它们并不原生支持的元素添加样式。Internet Explorer 8 及以下版本是个例外，不过第 (2) 步中描述了一个简单的解决办法。因此，跟着下面三个简单的步骤，就可以开始为包含 HTML5 元素的页面添加样式了。

针对全部浏览器为 HTML5 新元素添加样式

(1) 将图 11.3.1 中的代码添加到网站的主样式表文件（即所有页面都用到的样式表文件）。注意：如果使用 CSS 重置或 normalize.css（参见下一节）可以跳过这一步，它们会包含这里的代码。

(2) 有两种方式可以实现在 Internet Explorer 9 之前的 IE 中为新的 HTML5 元素设置样式。（只能选择一种，不能两种都用。）它们都使用了 HTML5 shiv 这个 JavaScript 文件（参见补充材料“关于 HTML5 shiv”）。

在这两种方法中，相对简单的一种是在每个页面的 head 元素（注意不是 header 元素）中添加图 11.3.2 中突出显示的代码。这

段代码会从外部网站（googlecode.com）加载该 JavaScript 文件。

更好的方式是将该 JavaScript 文件放在你自己的网站上。这是推荐的做法。这样可以避免在 googlecode.com 上的文件出问题时（尽管可能性不大，但仍有可能），网站不会在旧版本的 IE 中崩溃。可以通过 https://github.com/aFarkas/html5shiv/ 下载 HTML5 shiv 的 ZIP 文件（如图 11.3.3 所示），该地址是存放和维护 HTML5 shiv 代码的地方，再将它放入你的网站（如图 11.3.4 和图 11.3.5 所示）。

```
article, aside, figcaption, figure, footer,
→ header, main, nav, section {
    display: block;
}
```

图 11.3.1　大多数浏览器默认将它们无法识别的元素作为行内元素处理。因此这一小段 CSS 将强制 HTML5 新语义元素显示在单独的行。各浏览器内置的默认样式表均对 div、blockquote、p 等元素声明了 display: block;。更多详细信息参见 11.6 节

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>Le Journal</title>
<link rel="stylesheet"
→ href="css/lejournal.css" />
<!--[if lt IE 9]>
    <script src="http://html5shiv.
    → googlecode.com/svn/trunk/
    → html5.js"></script>
<![endif]-->
</head>
<body>
...
```

图 11.3.2　script 元素里面是一段所谓的条件注释（conditional comment）。将它放在引用样式表的 link 元素后面。[if lt IE 9] 开头的这段代码表示只有 Internet Explorer 9 以下版本的 IE 才加载这个 JavaScript 文件。通常，最好在页面结束的位置加载 JavaScript（原因见第 19 章）。不过，HTML5 shiv 是少有的必须在 head 中加载 JavaScript 的情况之一。否则，它就不起作用。

(3) 现在，可以随意使用 CSS 添加样式了！



图 11.3.3　将 HTML5 shiv 的 ZIP 文件保存到你的电脑，然后解压缩，得到将会使用的 JavaScript 文件

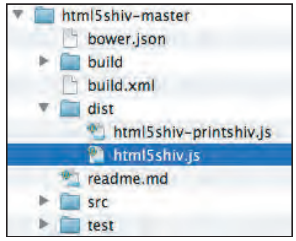


图 11.3.4　在电脑上打开该 ZIP 文件时，会发现其中包含了几个文件夹（如图所示）。你只需要关注 dist 这个文件夹，其中包含了将要用到的 html5shiv.js。将该文件复制到你的网站的某个子目录（如名为 js 的目录）里面。此外，还可以使用 html5shiv-printshiv.js。它与 html5shiv.js 基本相同，只是它还可以让 IE8 及更早版本的 IE 在打印时能够正确地处理新的 HTML5 元素

```
...
<link rel="stylesheet"
→ href="css/lejournal.css" />
<!--[if lt IE 9]>
    <script src="js/html5shiv.js">
    → </script>
<![endif]-->
</head>
<body>
...
```

图 11.3.5　突出显示的代码显示在网页的 head 中，引用样式表的 link 元素之后。这里与图 11.3.2 的唯一区别是 src 的值。该值应该引用你复制到网站里的 JavaScript 文件的位置。这段代码假定你将该文件放置在 HTML 页面所在文件夹下的 js 文件夹里（关于 URL，参见 1.7 节）。可以根据需要修改 src 里的路径，从而与个人网站的实际情况相符

关于 HTML5 shiv

与其他主流浏览器不同，Internet Explorer 8 及之前的版本会**忽略**它们不原生支持的元素的 CSS。HTML5 shiv 是专门用于解决这一问题的一段 JavaScript。（有时也称作 HTML5 shim。）Sjoerd Visscher 发起了一个研究 IE 特性的 Web 社区，HTML5 shiv 就是该社区的几个成员创建的。

HTML5 剃刀也集成进了其他一些 JavaScript 库，如 Modernizr（www.modernizr.com/）。因此，如果你在页面中使用了 Modernizr，就不需要单独加载 HTML5 剃刀了。顺便说一下，可以跳过前面步骤中的第 (2) 步。Modernizr 是一个非常方便的库，它可以检测浏览器是否支持 HTML5 和 CSS3 的各种特性。

11.4　对默认样式进行重置或标准化

前面提到，每个浏览器都有内置的默认样式表。HTML 会遵照该样式表显示，除非你自己编写的 CSS 覆盖了它们。整体上，不同浏览器提供的默认样式表是相似的，但也存在一定的差异。为此，开发人员在应用他们自己的 CSS 之前，常常需要抹平这些差异。

抹平差异的方法主要有两种（只使用其中一种即可）。

- 使用 CSS 重置（reset）开始主样式表，如 Eric Meyer 创建的 Meyer 重置（http://meyerweb.com/eric/tools/css/reset/）。另外还有其他的一些重置样式表。
- 使用 Nicolas Gallagher 和 Jonathan Neal 创建的 normalize.css 开始主样式表。该样式表位于 http://necolas.github.com/normalize.css/（最新版本使用 Download 按钮）。

从上面合适的 URL 中复制 CSS，并粘贴到你自己的样式表中。

CSS 重置可以有效地将所有默认样式都设为“零”，如图 11.4.1 所示。第二种方法，即 normalize.css，则采取了不同的方式。它并非对所有样式进行重置，而是对默认样式进行微调，这样它们在不同的浏览器中具有相似的外观，参见图 11.4.2。（重要提醒：我删除了前面两个图中的大尺寸图像，以便你能看到更多文本，重置或者 normalize.css 都不会隐藏图像。）

并非一定要用到这两种方法中的一种。保留浏览器的默认样式，并自己编写相应的 CSS 也不错。

如果确实要使用 normalize.css 或 CSS 重置，也不一定要保留它们提供的所有 CSS。它们的样式规则中，有一些所针对的 HTML 元素是你的网站并未使用的。这时，就不必在你的样式表中包含这些多余的 CSS。

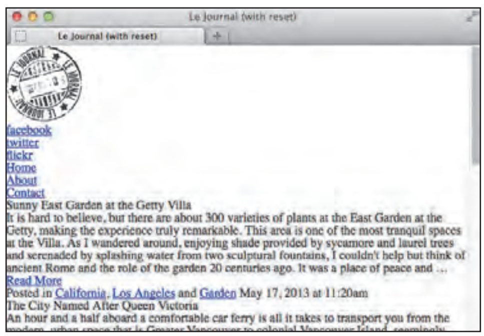


图 11.4.1　这是应用了重置的示例页面。最大的变化是所有的字体大小都变成一样的了，文本边框和列表样式没了，所有的外边距和内边距都设为了 0

在本章中，我使用了 normalize.css 中的一部分代码（如图 11.4.3 所示），并对文本添加了一些样式，形成一个初始的页面。在根据本章的介绍继续添加样式之前，页面的样子如图 11.4.4 所示。可以通过 www.htmlcssvqs.com/8ed/11 找到图 11.4.1、图 11.4.2 和图 11.4.4 对应的完整页面。



图 11.4.2　这是使用了 normalize.css（而不是重置）的示例页面。它与无样式的默认显示效果很相似，但也有差异。更重要的是，这个版本与使用当今常见的浏览器查看的效果是极为相似的

```
/*! normalize.css v2.1.2 | MIT License |
→ git.io/normalize */

article, aside, figcaption, figure, footer,
→ header, main, nav, section {
    display: block;
}

html {
    -ms-text-size-adjust: 100%;
    -webkit-text-size-adjust: 100%;
}

body {
    margin: 0;
}

a:focus {
    outline: thin dotted;
}

a:active,
a:hover {
    outline: 0;
}

small {
    font-size: 80%;
}

img {
    border: 0;
}
```

图 11.4.3　这是将 normalize.css 中本站不需要的样式规则移除后的一个版本



图 11.4.4　这是经过重置并定义好了文本格式后的示例页面的样子。在本章中，你将以此为起点对页面其余的样式进行定义

11.5　盒模型

CSS 处理网页时，它认为每个元素都包含在一个不可见的盒子里。这就是众所周知的**盒模型**，这里的盒子由内容区域、内容区域周围的空间（内边距，padding）、内边距的外边缘（边框，border）和边框外面将元素与相邻元素隔开的不可见区域（外边距，margin）构成，参见图 11.5.1。这类似于挂在墙上的带框架的画，其中图画是内容，衬边是内边距，框架是边框，而该画框与相邻画框之间的距离则是外边距。

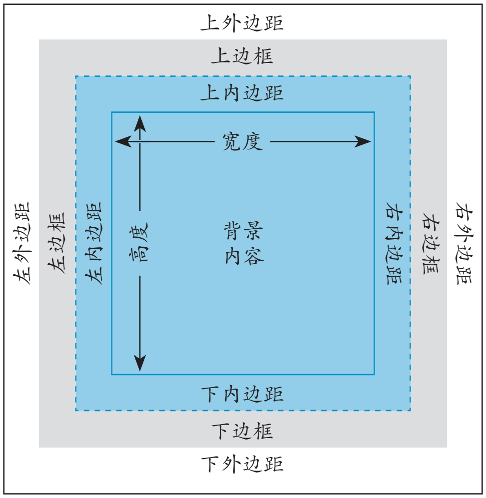


图 11.5.1　每个元素的盒子都有一些决定该元素所占空间及其外观的要素。可以使用 CSS 分别控制各个要素。注意，在默认情况下，宽度和高度仅定义内容区域的尺寸。背景（蓝色区域）会延伸到边框的后面，因此，通常情况下，它仅在内边距所延伸到的区域可见，除非边框是透明的或半透明的

可以使用 CSS 确定每个元素的盒子的外观和位置，并由此控制网页的布局，参见图 11.5.2。本章会深入讲解 width、padding、border、margin 和控制盒子外观的其他属性。但首先，理解盒模型的基础知识非常重要。

宽度、高度和盒模型

关于 CSS 的 width 属性对元素所显示宽度的影响，有两种处理方式。（不包含任何将其与邻近元素隔开的外边距。）

默认的处理方式（如图 11.5.1 所示）实际上有些有悖于常理。浏览器中元素的宽度与其 width 属性值并不一致（除非它没有内边距和边框）。CSS 中的宽度指示的是内边距里内容区域的宽度，如图 11.5.1 表示的那样。而元素在浏览器中的显示宽度则是内容宽度、左右内边距和左右边框的总和。显示高度与之类似，只不过计算的是上下内边距和边框值。



图 11.5.2　示例页面中附注栏的盒模型。记住，每个元素都有自己的盒子。例如，这里显示的 img 元素有边框，而它左右的空间则是包含该图像的 article 元素的内边距

对大多数代码编写人员来说，另一种方式则更为直观。使用这种方式的话，元素的显示宽度就等于 width 属性的值。内容宽度、内边距和边框都包含在里面（如图 11.5.3 所示）。height 属性与之类似。要使用这种模式，仅需对元素设置 box-sizing: border-box;。

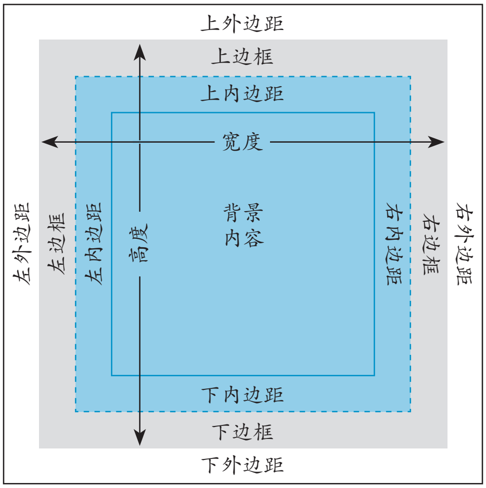


图 11.5.3　如果设置了 box-sizing: border-box;，则宽度和高度就包含了除外边距以外的所有要素。如果不设置这条规则，则表现为图 11.5.1 所示的方式

以上讲述都比较抽象，因此不妨看看分别使用两种方式的一个实例（如图 11.5.4 所示）。注意这里显示的 padding 和 border-width 值在每个边都起作用。例如，padding: 15px; 会在水平方向总共产生 30 像素的内边距（左 15 像素，右 15 像素），垂直方向也是 30 像素（上 15 元素，下 15 像素）。

提示　图 11.5.1 是根据 Rich Hauck 的盒模型图绘制的（该图本身是根据 CSS 规范中的插图绘制的）。

提示　关于 box-sizing: border-box;、通常不设置 height 值的原因等主题，参见 11.7 节。

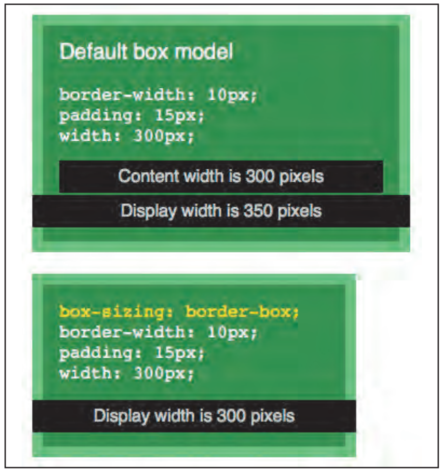


图 11.5.4　两个 div 的 border-width（浅绿色）、padding 和 width 值都相同。上图是以默认盒模型（如图 11.5.1 所示）显示的，下图则是以设置了 box-sizing: border-box; 的盒模型（如图 11.5.3 所示）显示的，其显示宽度与 width 属性值相同（都是 300 像素）。两个盒子都设置了 height: 170px;（这里没有显示出来），不过第一个要更高一些，这是由于上下内边距和边框尺寸增加了显示高度（注意，黑色的横条不是这些 div 的一部分，因此它们不会影响 div 的高度）

11.6　控制元素的显示类型和可见性

正如第 1 章讲到的，每个元素在默认情况下要么显示在单独的行（如 h1 ~ h6、p 等），要么显示在行内（如 em、strong、cite 等），如图 11.6.1 所示。第 1 章还提到，前一种元素称为块级元素，后一种称为行内元素（这是一种非官方的叫法）。

造成这种情况的本质是它们的 display 属性，即块级元素被设置为 display: block（对于 li 元素为 display: list-item），而行内元素被设置为 display: inline。

当然，使用 CSS 可以改变元素的默认显示类型，例如将 block 改为 inline（如

图 11.6.2 所示），或者反过来（如图 11.6.3 所示）。还有一种混合显示方式称为 inline-block，让元素与其他内容出现在同一行，同时具有块级元素的属性（如图 11.6.4 ～图 11.6.6 所示）。还可以通过设置 display 属性让元素完全不在页面上显示（如图 11.6.7 所示）。

```
<p>This is the <em>first</em> paragraph.</p>
<p>This is the <em>second</em> paragraph.</p>
```

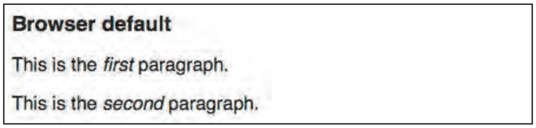


图 11.6.1　这些简单嵌套 em 元素的段落演示了 display: block 和 display: inline 的区别

```
p {
    display: inline;
}
```

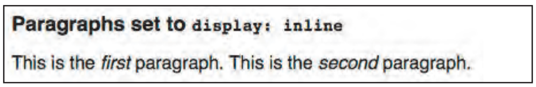


图 11.6.2　两个段落看起来像一个段落

```
em {
    display: block;
}
```

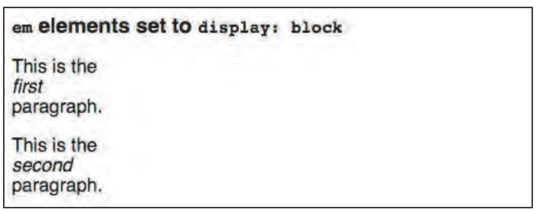


图 11.6.3　em 元素显示在单独的行，就像段落一样

```
em {
    background: lightgreen;
    display: inline-block;
    width: 300px;
}
```

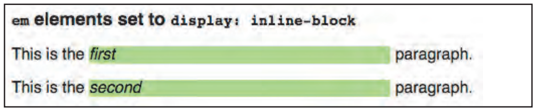


图 11.6.4　设置为 inline 的元素会忽略任何 width、height、margin-top 和 margin-bottom 设置。不过，设置为 inline-block 的元素可以使用这些属性，如这里的 em 可以设置 width。因此，如果本例对 em 设置 display: inline;，则会像图 11.6.1 那样显示（除了背景色），无论是否对它指定 width（所有的 display 类型都接受 background 样式。这里将 em 的背景设为浅绿色，是为了让 em 的宽度变得明显）

```
.social-sites li {
    display: inline;
}

.nav-main li {
    display: inline-block;
}

.nav-main a {
    color: #292929;
    display: inline-block;
    font-family: 'Open Sans', sans-serif;
    font-size: 1.125em;
    font-weight: 700;
    text-transform: uppercase;
}
```

图 11.6.5　这里将主导航链接及其父元素列表项目设置为 inline-block，从而方便为其添加上下内边距（参见 11.8 节）。社交网站图标（指这里的 .social-sites li 选择的元素）不需要添加块级元素才具有的样式，因此将它们设置为 inline 就够了

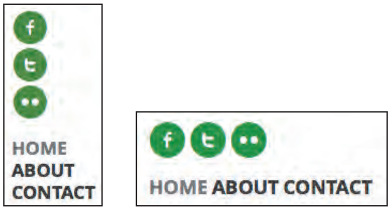


图 11.6.6 这里是一个演示 display 属性功能的实例。社交网站图标与主导航都包含在列表项目（li）里，因此，在默认情况下，它们从上往下排列（如左图所示）。将它们默认的 display: list-item; 覆盖后（如图 11.6.5 所示），每个链接列表中的元素都显示在一行内

```
em {
    display: none;
}
```

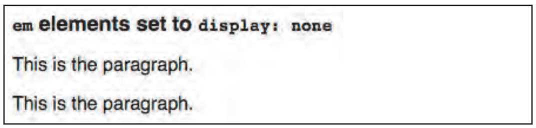


图 11.6.7 em 文字既不显示，也不占据任何视觉空间。在 HTML 中紧跟在它后面的文字会向左移动，完全无法察觉到 em 文字曾占据那个位置

文档流

默认情况下，元素会按照它们在 HTML 中自上而下出现的次序显示（这称为**文档流**，document flow），并在每个非行内元素的开头和结尾处换行。对于这一点，图 11.2.1 和图 11.2.2 提供了一个比上面这些例子更长的例子。

1. 指定元素的显示方式

(1) 在样式表规则中，输入 display:。

(2) 输入 block，让元素显示为块级元素（就像开始新的段落），参见图 11.6.3；

或者输入 inline，让元素显示为行内元素（不同于开始新的段落），参见图 11.6.2；

或者输入 inline-block，让元素显示为行内元素，同时具有块级元素的特征，即可以为元素的四条边设置 width、height、margin、padding 等属性，参见图 11.6.4 ～图 11.6.6；

或者输入 none，隐藏元素，并将其从文档流中完全移除（参见图 11.6.7）。

关于 display 属性的其他值，参见提示。

2. 控制元素可见性

visibility 属性的主要目的是控制元素是否可见。与 display 属性不同的是，使用 visibility 隐藏元素时，元素及其内容应该出现的位置会留下一片空白区域（如图 11.6.8 所示）。隐藏元素的空白区域仍然会在文档流中占据位置。

(1) 在样式表规则中，输入 visibility:。

(2) 输入 hidden，让元素不可见，但在文档流中保留它（如图 11.6.8 所示）；

或者输入 visible，让元素显示出来。

```
em {
    visibility: hidden;
}
```

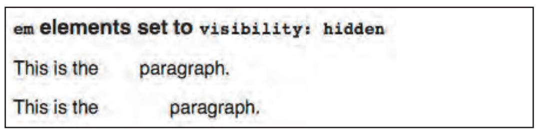


图 11.6.8 隐藏的 em 文字所在的位置出现了一块空白，就像是将 em 文字设成了白色。空白的尺寸与文本（或者其他任何隐藏的内容）的大小完全一致

提示 设置为 display: inline; 的元素不接受 padding 的设置，但 padding-top 和 padding-bottom 会越界进入相邻元素的区域，而不是局限于该元素本身的空间。可以对元素设置 background: red; 以作测试。

提示 第 12 章演示了一个针对窄屏（如手机屏幕）将行内元素设置为 display: block;，针对宽屏（如桌面计算机的屏幕）改回 display: inline; 的实例。

提示 设置为 display: none; 或 visibility: hidden; 的元素里的所有内容（包括所有的后代）也会受到影响。例如，如果将一个包含一些 p、figure 和 img 元素的 article 元素设置为 display: none;，则这些元素都不会显示出来。如果将 article 元素设置为 visibility: hidden;，就会出现一片空白（很有可能是一大块空白）。

提示 display 属性还有其他几个值，但浏览器对它们的支持程度是有差异的。例如，grid 和 flex 属性提供了一些额外的布局技术，但它们还没有进入最终的规范。更多信息参见 http://developer.mozilla.org/CSS/display。

提示 除了 inherit，visibility 属性还有一个值：collapse，用于 table 元素的部分内容。关于 collapse 的更多信息，参见 http://developer.mozilla.org/CSS/visibility。

11.7　设置元素的高度和宽度

可以为很多元素设置高度和宽度，如分块内容、段落、列表项、div、图像、video、表单元素等（参见图 11.7.1 ～图 11.7.3）。

同时，可以为短语内容元素（默认以行内方式显示）设置 display: block; 或 display: inline-block;，再对它们设置宽度或高度。（关于 display 属性的更多信息，参见 11.6 节。）

```
main {
    width: 600px; /* 62.5% = 600px/960px */
}

.sidebar {
    width: 300px; /* 31.25% = 300px/960px */
}
```

图 11.7.1　最终，页面会有两栏，一栏是 main 里的内容，一栏是 class="sidebar" 的 div 里的内容。这些规则分别对它们设置了固定的宽度（如图 11.7.2 和图 11.7.3 所示）



图 11.7.2　这里将 main 的宽度设置为 600px 以适应图像的宽度。现在，无论浏览器窗口的宽度是多少，页面主区域的宽度会始终保持 600px 不变。文字会自动折行，不会超过图像的宽度

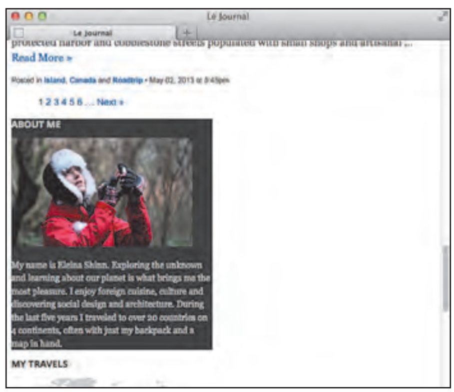


图 11.7.3　侧边栏以 About Me 模块开头。目前，我们的页面只有一列，可以看到 main 内容的尾部位于侧边栏的上方。还可以看到，侧边栏的宽度（目前设置为 300px）只有 main 的宽度的一半

1. 设置元素高度或宽度的步骤

(1) 输入 width: *w*，其中的 *w* 是元素内容区域的宽度，可以表示为长度（带单位，如 px、em 等）或父元素的百分数。或者使用 auto 让浏览器计算宽度（这是默认值）。

(2) 输入 height: *h*，其中的 *h* 是元素内容区域的高度，只能表示为长度（带单位，如 px、em 等）。或者使用 auto 让浏览器计算高度（这是默认值）。

对定宽页面使用像素，对响应式 Web 设计使用百分数

本章在介绍网页布局方法时，对宽度使用像素为单位。在代码注释里，说明了与像素宽度等价的百分数，以及如何算出该数值的（如图 11.7.1）。在第 12 章介绍响应式布局时，则会使用百分数，而不是像素。

提示 内边距、边框和外边距都不包含在宽度或高度的数值里（参见 11.5 节及本节后面的“宽度、外边距和 auto”部分）。11.5 节还介绍了对元素设置 box-sizing: border-box;（如图 11.7.4 所示），让宽度和高度包含内边距和边框的方法。

```
* {
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
}
```

图 11.7.4 可以使用 * 通配符对所有元素应用 border-box 的规则。当然，也可以单独对元素应用该规则（用你需要的选择器替换 * 即可）。带有 -webkit- 和 -moz- 这些奇怪前缀的属性可以让这些规则在旧的 Android 和 iOS 设备上起作用，同时在 Firefox 上也能正常工作。

提示 百分数是相对于父元素的宽度的，而不是相对于元素本身的原始宽度的。例如，假设 div 宽度为 100 像素，将其内部的某个元素的宽度设为 70%，则这个子元素的宽度为 70 像素，你在下一章还能看到这方面的例子。

提示 不能为显示为行内元素的元素（如短语内容）设置高度或宽度，除非为它们设置 display: inline-block 或 display: block。关于 display 属性的更多信息，参见 11.6 节。

提示 还有 min-width、min-height、max-width 和 max-height 属性用来设置最小和最大尺寸。max-width 属性是为流式布局（下一章会讲到一个流式布局的例子）设置宽度限制的绝佳工具。在那个例子中，对包围整个页面的 .page div 设置了 max-width: 960px;。同时，main 和 .sidebar 均使用百分数宽度，因此页面在小一些的屏幕上会较窄，在大一些的屏幕上则较宽，但不会超过 960 像素，哪怕屏幕非常大。另外，参考补充材料“为什么 min-height 通常比 height 更适用”。

提示 width 和 height 不是继承的。

2. 宽度、外边距和 auto

如果不显式设置宽度和高度，浏览器就会使用 auto。对于大多数默认显示为块级元素的元素，width 的 auto 值是由**包含块**的宽度减去元素的内边距、边框和外边距计算出来的，参见图 11.7.5 和图 11.7.6。简单说来，包含块的宽度指的是父元素给元素留出的宽度。

```
div { /* 包含块 */
     background: khaki; /* tan */
     width: 300px;
}

p,
.second {
     background: white;
     border: 3px solid royalblue;
     margin: 10px;
     padding: 8px;
}

.second { /* 第二个段落 */
     border-color: orangered;
     width: 200px;
}
```

图 11.7.5　在这个例子中，将父元素 div 的 width 设为 300 像素。这将是我们的包含块。然后，两个段落各个边上都有 10 像素的外边距、8 像素的内边距及 3 像素宽的边框。第一个段落的宽度是自动设置的，因为 auto 是 width 的默认值（除非指定其他的值）。第二个段落（在 HTML 中带有 class="second"）设为 200px

例如，位于 200 像素宽的空间里的 p 元素的 auto 宽度为 200 像素。但如果为其设置 padding-right: 10px，则 auto 宽度为 190 像素。

对于图像这样的元素，auto 宽度等于它们固有的宽度，即外部文件的实际尺寸（如本章示例页面中最大的图像，大小为 600 × 365）。行内元素会完全忽略 width 属性（即不能为 em、cite 等元素设置宽度，除非将它们设置为 display: inline-block 或 display: block）。关于 display，参见 11.6 节。

如果手动设置 width、margin-left 和 margin-right 值，但这些值加上边框和内边距的值不等于包含块的大小，有的属性就需要作出让步。实际上，浏览器会优先考虑 width，确保右边的间距大于你设置的 margin-right（参见图 11.7.6 中底部的段落）。

如果手动设置 width，但将某个外边距设为 auto，那么这个外边距将进行伸缩以弥补不足的部分。

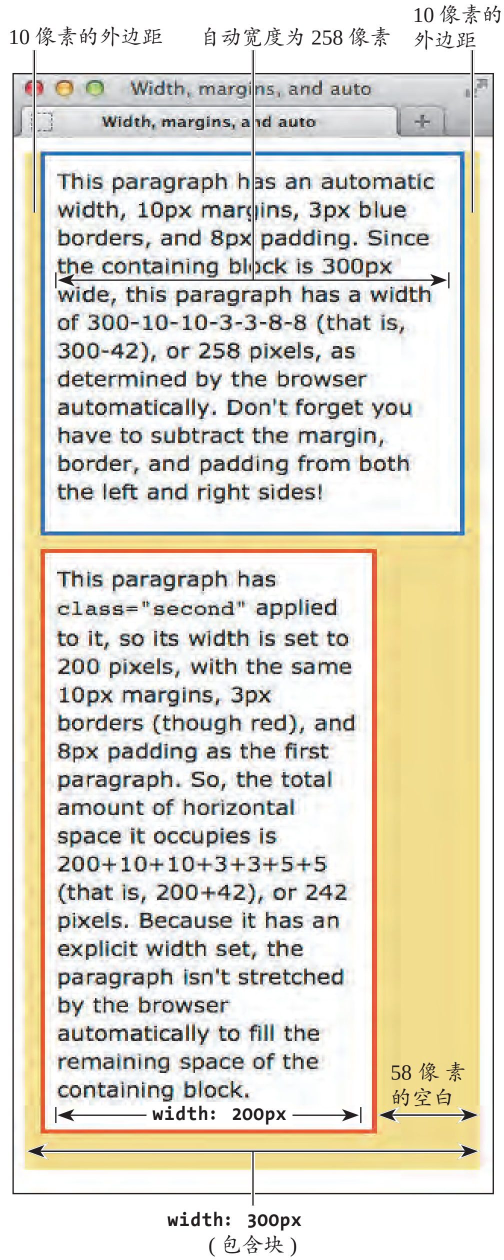


图 11.7.6　如果 width 是 auto（就像第一个段落那样），它的值就是由包含块（黄褐色区域）的宽度减去元素自身的外边距、内边距和边框计算出来的。如果 width 是手动设置的（就像最后一个段落那样），则右外边距常常会进行调整以填补不足的空间

不过，如果手动设置 width，并将左右外边距都设为 auto，那么两个外边距就将设为相等的最大值，这样会导致元素居中。例如，.page { margin: 0 auto; } 会使对应的元素在页面居中显示。这也是我对示例页面所做的设置，参见图 11.7.6 和图 11.10.7。

为什么 min-height 通常比 height 更适用

除非你确定元素的内容不会变得更高，最好避免在样式表中指定高度。在大多数情况下，可以让内容和浏览器自动控制高度。这可以让内容在浏览器和设备中根据需要进行流动。

如果设置了高度，随着内容变多，它们有可能撑破元素的盒子，这可能是你预期之外的。在这种情况下，符合标准的浏览器**不会**自动扩大高度，它们在你指定高度时采用了这个指令，并一直坚持下去。（了解相关示例请参考 11.16 节。）

不过，如果希望元素**至少**具有某一特定高度，可以设置 min-height。如果内容日后变多，元素的高度会自动按需增长。这是 height 与 min-height 的区别，width 和 min-width 也是类似的。

或许你还有所疑虑，实际上，很多原因都会导致内容增长。你的内容或许来自数据库、第三方源或由用户生成的内容。同时，访问者可能增大其浏览器的字体大小，覆盖你指定的样式。

11.8　在元素周围添加内边距

顾名思义，内边距就是元素内容周围、边框以内的空间。根据上文的类比，内边距就像是图画（内容）与画框（边框）之间的衬边。可以改变内边距的厚度（如图 11.8.1 到图 11.8.4 所示），不能改变它的颜色或纹理。显示在内边距区域的颜色和纹理是元素的背景，是通过 background、background-color（参见图 11.8.1）或者 background-image 设置的。

```
.about {
    background-color: #2b2b2b;
    padding: .3125em .625em .625em;
}

/*
跟padding: 5px 10px 10px;
是一样的，因为字体大小是16px，且
.3125 = 5px/16px
.625 = 10px/16px
*/
```

图 11.8.1　同外边距类似，如果为 padding 设置四个值，那么它们分别表示上、右、下、左（按时钟顺序）内边距。因此在这里四个边都有内边距（实现结果如图 11.8.2 所示）



图 11.8.2　如果不设置内边距（左），内容就会贴到边界。添加内边距后，元素里的内容外围就会产生一些空白（右）。同时，在内边距的区域会显示背景色。（如果添加 .about { border: 1px solid red; }，则整个盒子的外围会显示一条红色的边框，边框位于内边距的外面。）可以看到，这个模块的上边和下边的空白要大于内边距指定的大小。这是由于浏览器为标题（ABOUT ME）设置的默认上外边距和为段落（如图像下面的段落）设置的默认下外边距造成的。它们外围的 .about 盒子的内边距让这些外边距在盒子内部生效

```
.nav-main a {
    color: #292929;
    display: inline-block;
    font-family: 'Open Sans', sans-serif;
    font-size: 1.125em; /* 18px/16px */
    font-weight: 700;
    padding: .5em 1.15em .5em 1.4em;
    text-transform: uppercase;
}

/*
跟padding: 9px 20.7px 9px 25.2px;
一样，因为字体大小为18px，且
.5   = 9px/18px
1.15 = 20.7px/18px
1.4  = 25.2px/18px
*/
```

图 11.8.3　现在，我们为主导航里的每个链接都添加了内边距。在下一部分为其添加边框之后，上下内边距会更加明显

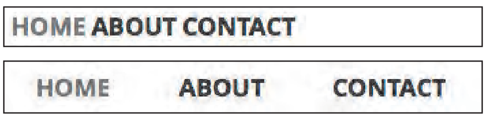


图 11.8.4　这些链接之前都挤在一起（上），添加了内边距后，它们就有“呼吸”的空间了（下）

padding 的简记法

同 border 和 margin 属性一样，padding 也可以使用简记法，从而不必使用 padding-top、padding-right 等属性为每个边都单独设定内边距。

- ❑ padding: 5px;——使用一个值，这个值就会应用于全部四个边。
- ❑ padding: 5px 9px;——使用两个值，则前一个值会应用于上下两边，后一个值会应用于左右两边。
- ❑ padding: 5px 9px 11px;——使用三个值，则第一个值会应用于上边，第二个值会应用于左右两边，第三个值会应用于下边（如图 11.8.1 和图 11.8.2 所示）。
- ❑ padding: 5px 9px 11px 0;——使用四个值，它们会按照时钟顺序，依次应用于上、右、下、左四个边（如图 11.8.3 和图 11.8.4 所示）。

内边距的值可以使用像素、百分数、em 或 rem 的组合。

在元素周围添加内边距的方法

输入 padding: *x*;，这里的 *x* 是要添加的空间量，表示为带单位（通常为 em 或像素）的长度或父元素宽度的百分数（如 20%）。

也可以输入 padding-top: *x*;、padding-right: *x*;、padding-bottom: *x*; 或者 padding-left: *x*; 单独为一个边添加内边距，参见图 11.8.5 和图 11.8.6。

```
.links {
    padding-left: 0;
    width: 270px; /* 90% = 270px/300px */
}
```

图 11.8.5　侧边栏包含两组链接，每组都包含在一个带 class="links" 的无序列表（ul）里。这里将内边距设为 0，以覆盖浏览器默认的 40px（如图 11.8.6 所示）。宽度的样式能确保链接在距侧边栏（先前设定为 300 像素宽）右侧 30 像素的地方折行（关于列表，参见第 15 章；关于列表的默认左内边距，重点参考 15.3 节）

提示　关于如何确定图 11.8.1 和图 11.8.3 中的 em 值的信息参见 11.10 节补充材料。



图 11.8.6　列表默认的缩进样式很难看（上），现在，由于取消了内边距，链接与标题对齐了（下）

提 示　默认情况下，元素的 width 和 height 不包含 padding 的大小。参见前一节的“宽度、外边距和 auto”部分及 11.5 节（其中还讲解了如何覆盖默认的盒模型设置）。

提 示　关于背景样式的设置，参见 10.10 节、14.6 节和 14.7 节。

提 示　内边距不是继承的。

11.9　设置边框

可以在元素周围创建边框（参见图 11.9.1 和图 11.9.2），或针对元素的某一边设置边框（参见图 11.9.3 ~ 图 11.9.10），并设置它的厚度、风格和颜色。如果指定了内边距（参见 11.8 节），边框将包围在元素内容和内边距的外面（参见图 11.9.5 ~ 图 11.9.10）。图 11.9.11 和图 11.9.12 展示了我们可用的所有边框样式。

```
.about img {
    border: 5px solid #bebebe;
}
```

图 11.9.1　边框可以应用于任何元素，包括图像。如果希望对一个以上的元素设置相同的边框样式，最好引入一个类，从而可以复用，例如 .frame { border: 5px solid #bebebe; }

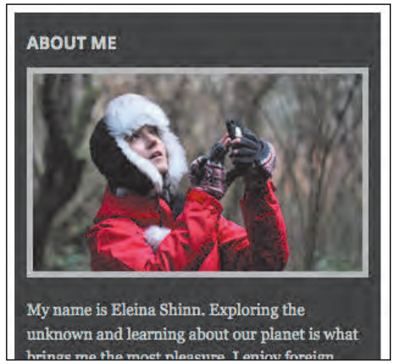


图 11.9.2　边框将图像很好地框起来了

```
.nav-main {
    /* 绿色 */
    border-top: 5px solid #019443;
    /* 灰色 */
    border-bottom: 1px solid #c8c8c8;
}

.nav-main li {
    border-left: 1px solid #c8c8c8;
    display: inline-block;
}

.nav-main li:first-child {
    border-left: none;
}
```

图 11.9.3　这些简单的边框对页面的主导航和头部起到了下定义的作用（如图 11.9.4 所示）。作为一种常见的做法，每个导航链接都包含在一个列表项目（li）里。注意，这里对第一个 li 使用了 border-left: none;，以取消其左边框（即改回至它的默认样式）

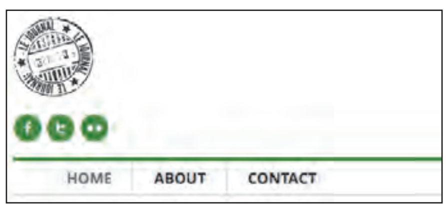


图 11.9.4　边框让主导航更加突出，让导航链接彼此分开，还能有效地区分页面头部和它下面的内容（这里没有显示出来）。水平的边框会延伸至页面的整个宽度

```
.nav-main {
    padding: .45em 0 .5em; /* 7px 0 8px */
}
```

图 11.9.5　导航容器顶部和底部的一点点儿内边距使格局发生了微妙的变化



图 11.9.6　内边距将水平线和垂直线分开了

1. 定义边框风格

输入 border-style: *type*，其中的 *type* 可以是 none、dotted（点线）、dashed（虚线）、solid（实线）、double（双线）、groove（槽线）、ridge（脊线）、inset（凹边）或 outset（凸边）。

2. 设置边框宽度

输入 border-width: *n*，这里的 *n* 是需要的宽度（含单位，如 4px）。

3. 设置边框颜色

输入 border-color: *color*，这里的 *color* 是颜色名称、十六进制值或 RGB、HSL、RGBA、HSLA 颜色（参见 7.5 节的"CSS 颜色"）。

4. 使用简记法同时设置多个边框属性

(1) 输入 border。

(2) 如果需要，输入 -top、-right、-bottom 或 -left 将边框效果限制在某一条边上。

(3) 如果需要，输入 *-property*，这里的 *property* 可以是 style（风格）、width（宽度）或 color（颜色），仅使用单个属性设置边框。

(4) 输入冒号（:）。

(5) 输入恰当的值（如前面介绍的那样）。如果跳过了第 (3) 步，则可以指定所有三种边框属性（如 border:1px solid 和 border-right: 2px dashed green;）。如果在第 (3) 步中指定了一种属性，则只能使用这种属性可以接受的值（如 border-right-style: dotted;）。

提示　默认情况下，Internet Explorer 会对作为链接的图像添加一个蓝色的边框。为了取消这一样式，可以在样式表中添加 img { border: none; }。这里，示例页面也加入了这一规则，因为网站的标识就位于一个指向主页的链接里面。（这个例子是主页，但整个网站的页面都使用相同的标识代码。）

提示　border 简写属性及各个边框属性（border-width、border-style 和 border-color）均可接受一至四个值。如果使用一个值，那么它会应用于全部四个边。如果使用两个值，那么前一个值会应用于上下两边，后一个值会应用于左右两边。如果使用三个值，那么第一个值会应用于上边，第二个值会应用于左右两边，第三个值会应用于下边。如果使用四个值，那么它们会按照时钟顺序，依次应用于上、右、下、左四个边。

提示　要让边框显示出来，至少必须定义边框样式。如果没有定义样式，就不会显示边框。边框样式的默认值是 border-style: none。

提示　如果使用简记法，如 border 或 border-left 等，则未提供的属性将显示其默认值。因此，border: 1px black; 相当于 border: 1px black none;，这意味着不会显示边框（即便在之前使用 border-style 指定了边框样式）。

提 示 边框的默认颜色是元素的 color 属性的值（参见 10.9 节）。

提 示 border 属性可用于表格及其单元格，相关示例参见 18.1 节。

提 示 默认情况下，元素的宽度和高度设置不包括边框大小。11.7 节第一条提示解释了如何覆盖默认设置。

提 示 CSS3 引入了 border-image 属性。除 Internet Explorer 以外，浏览器对它的支持程度较好（参见 http://caniuse.com/#search=border-image）。关于 border-image，参见 www.sitepoint.com/css3-border-image 和 css-tricks.com/。

提 示 边框不是继承的。

```
.about h2,
.mod h2 {
    font-size: 0.875em;
}

.about h2,
.mod h2,
.nav-main a {
    font-family: 'Open Sans',sans-serif;
    font-weight: 700;
    text-transform: uppercase;
}

.mod h2 {
    border-bottom: 1px solid #dbdbdb;
    padding-bottom: 0.75em;
}
```

图 11.9.7 我在一条声明中为多个标题添加了边框，效果参见图 11.9.8（名为 mod 的类代表了页面中的一个通用模块）

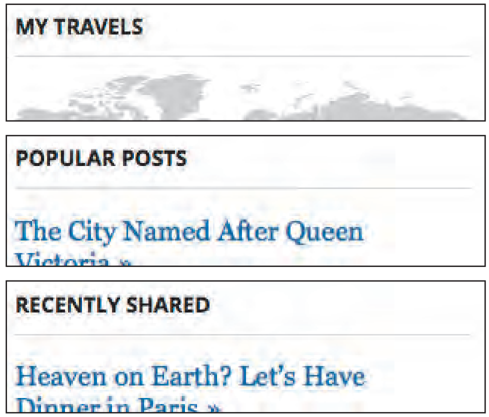


图 11.9.8 我们对附注栏中的三个标题也应用了同样的样式，底部的内边距确保了边框和标题文本之间拉开了一点距离

```
.footer p {
    font-size: .6875em; /* 11px/16px */
}

.post-footer {
    border-bottom: 1px solid #dbdbdb;
    border-top: 1px solid #dbdbdb;
    padding-bottom: .7em; /* 7.7px/11px */
    padding-top: .7em;
}
```

图 11.9.9 除了都有一个上边框，这里还对每篇博客文章下方的页脚应用了相似的效果（如图 11.9.8 所示）。每个这样的页脚都是一个带 class="post-footer" 的段落



图 11.9.10 如果没有上下内边框（如图 11.9.7 所示），则边框几乎会贴着文字（上）。添加了内边框后，效果看起来好多了（下）

11

```
p {
    border: 10px solid red;
    padding: 15px;
}

.ddd {
    border-width: 4px;
    border-style: dotted dashed double;
}

.ridge {
    border-style: ridge;
    border-color: orange;
}

.groove {
    border-style: groove;
    border-color: purple;
}

.inset {
    border-style: inset;
    border-color: blue;
}

.outset {
    border-style: outset;
    border-color: green;
}
```

图 11.9.11　在这个例子中，我为每个段落设置了内边距和默认边框。然后，对第一个段落设置了应用于四个边的边框宽度，并为每个边设置了不同的样式。对于余下的四个段落，使用一条声明并在其中重复 10px 比通过两个属性分开设置样式和颜色要方便一些

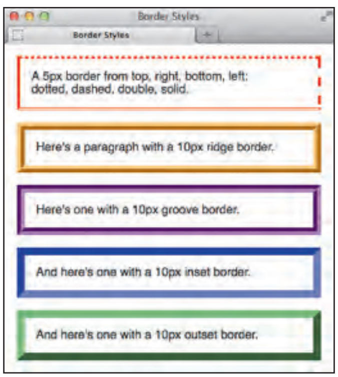


图 11.9.12　各浏览器对边框样式的处理方式并不完全相同。这是在 Firefox 中显示的不同边框样式，从中可以了解到不同样式的差异。另见彩插

11.10　设置元素周围的外边距

外边距是元素与相邻元素之间的透明空间（参见图 11.10.1 和图 11.10.2）。关于它与元素边框和内边距的关系，参见 11.5 节。

```
h1 {
    color: #212121;
    font-family: "Lato", sans-serif;
    font-size: 3.25em; /* 36px/16px */
    font-weight: 300;
    letter-spacing: -2px;
    line-height: .975;
    margin-bottom: .4125em; /* 21.45px */
}
```

图 11.10.1　我用自己的设置覆盖了默认的下外边距设置，关于使用 em 设置外边距的说明参见补充材料



图 11.10.2　博客条目标题和图像之间的默认空间太大了（上图），重新设置后两者更为紧凑了，看起来好很多（下图）

设置元素外边距的方法

输入 margin: *x*，其中的 *x* 是要添加的空间量，可以表示为长度、相对父元素宽度的百分数或 auto。

也可以输入margin-top: *x*;、margin-right: *x*;、margin-bottom: *x*;（参见图 11.10.1 和图 11.10.3）或者 margin-left: *x*; 为元素的一条边应用外边距。

```
.map {
    margin: 1.4375em 0 .8125em;
    /* 23px 0 13px */
}

.links {
    margin: 1.5em 0 4.125em;
    /* 24px 0 66px */
    padding: 0;
}

.links li {
    margin-bottom: 1.1em;
}
```

图 11.10.3　以下几条样式规则处理几个元素之间的间距，第一条规则为地图周围留出了一些空间，如图 11.10.4 所示，后面两条规则影响了附注栏中的链接，如图 11.10.5 所示



图 11.10.4　地图和两个标题之间显得很挤（左图），外边距解决了这一问题（右图）

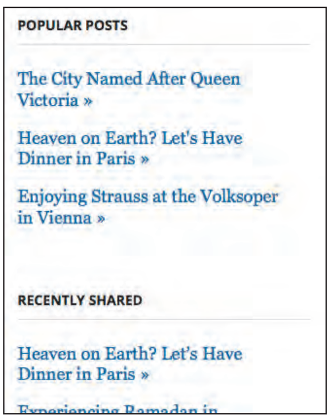


图 11.10.5　每组链接之间的空隙更大了，不同链接之间的空隙也变得更大了（关于之前的样子，参见 11.8 节的图 11.8.6）

提 示　如果对 margin 使用一个值，这个值就会应用于全部四个边。如果使用两个值（参见图 11.10.6），那么前一个值会应用于上下两边，后一个值会应用于左右两边。如果使用三个值（参见图 11.10.3），那么第一个值会应用于上边，第二个值会应用于左右两边，第三个值会应用于下边。如果使用四个值，那么它们会按照时钟顺序，依次应用于上、右、下、左四个边。

提 示　margin 属性的 auto 值依赖于 width 属性的值（如图 11.10.6 所示），参见 11.7 节。

提 示　如果元素位于另一个元素的上面，对于相互接触的两个 margin（即元素相互接触的下外边距和上外边距），仅使用其中较大的一个，另一个外边距会被叠加。左右外边距不叠加。

提 示　确定 em 值时，可能需要在手头准备一个计算器。此外，还可以使用 http://pxtoem.com 这样的工具，它们可以帮你计算 em 值。

提 示　外边距不是继承的。

```
.page {
    border: 1px solid red; /* 临时设置 */
    margin: 0 auto;
    width: 960px;
}
```

图 11.10.6　让网页的主体内容在浏览器里水平居中显示是一种常见的样式，要实现这种样式其实并不难。首先，对内容的容器添加 width（也可以用 max-width）。然后将其左右外边距都设为 auto。这样做的话，浏览器会基于浏览器宽度和容器宽度之差计算这些外边距。在本例中，带 class="page" 的 div 包含了整个页面。

11

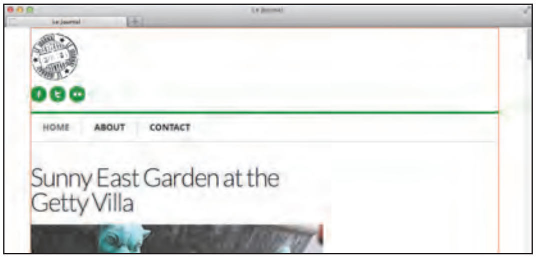


图 11.10.7　页面内的内容可能是左对齐的，但页面本身却位于浏览器窗口的中间。这里临时为 `.page div` 添加了一个红色的边框，好让效果更为明显。这里还将该 `div` 的宽度设为 960px，从而为侧边框腾出空间。我们将在下一节学习如果将附注栏放到那个位置

理解内边距和外边距的 em 值

当 em 值用于内边距和外边距时，它的值是相对于元素的字体大小的，而不是相对于父元素的字体大小的（你可能会这样认为），具体参见 10.6 节。计算外边距或内边距的 em 值的公式为：

要指定的字体大小／元素的字体大小＝值

设想一个简单的例子，其中的段落定义了 `p { font-size: 14px; padding: .5em; }`。那么，它在四个边的内边距都是 7 像素，因为 7/14 = 0.5。如果将字体大小改为 20px，内边距就会自动变成 10 像素（10/20 = 0.5）。这就是 em 这样的相对单位的作用——随着布局放大或缩小，布局的比例始终不变。

外边距也是类似的。如图 11.10.1 所示的，`h1` 的字体大小大约是 36 像素（3.25em = 36px/16px）。因此，0.4125em 的下外边距大约为 21.45 像素（21.45/36 = 0.4125）。

像 em、百分数以及 rem 这样的相对单位在响应式网页中的优势更为明显（响应式 Web 设计将在下一章进行介绍）。不过它们在固定宽度的布局中也有作用。Trent Walton 介绍了一个强大的使用相对单位的例子，见 http://trentwalton.com/2013/01/07/flexible-foundations。

对于内边距和外边距的设置，建议使用相对单位，就像本书的例子中演示的那样。我知道，在刚刚开始学习网站构建的时候，需要一段时间去习惯这种做法。这也是我既提供 em 值又提供像素值的原因。刚开始的时候，你可能会感觉对内边距和外边距使用像素更舒服一些。这样做没有问题。你可以在自己认为合适的时候转向 em。

这里需要指出一条简短的说明：如果要对内边距和外边距使用百分数，通常不会将它们用于上下边距的值，因为这样的值是基于其包含块的宽度的。

11.11　使元素浮动

可以通过 `float` 属性使元素浮动在文本或其他元素上。可以使用这种技术让文本环绕在图像（参见图 11.11.1 ～图 11.11.3）或者其他元素周围，也可以使用这种技术创建多栏布局等，如图 11.11.4 和图 11.11.5 所示。

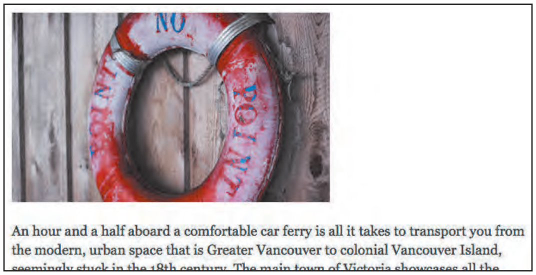


图 11.11.1　这是一篇博客文章的片段。默认情况下，段落显示在图像的下面。段落没有设置宽度，文字折行是因为段落（以及图像）位于 main 元素中，而该元素在先前设置了 600 像素的宽度（又如图 11.11.3 所示）。图像的宽度（370 像素）是在 HTML 中指定的。如何让文字围绕在图像周围呢？

```
.post-photo {
    float: left;
    margin-bottom: 2px;
    margin-right: 22px;
}
```

图 11.11.2　这很简单！只要让一个设置了宽度的元素浮动，在正常情况下显示在它下面的内容就会流动到它的周围（前提是这些内容没有设置宽度）。这里的图像（带有 class="post-photo"）和段落就满足这些条件，因此会形成如图 11.11.3 所示的效果。外边距的设置让图像和文字之间有了一些空隙



图 11.11.3　图像向左浮动时，如果文本的高度大于图像的高度，文本会环绕在图像周围。这个例子使用的是图像，但你也可以使用同样的方法让 figure、aside 或其他元素浮动

```
main {
    float: left;
    width: 600px; /* 62.5% = 600px/960px */
}

.sidebar {
    float: right;
    margin-top: 1.875em; /* 30px/16px */
    width: 300px; /* 31.25% = 300px/960px */
}
```

图 11.11.4　可以用类似的方法让两个元素并排在一起，如主要内容和附注栏。它们都指定了宽度，因此都可以进行浮动

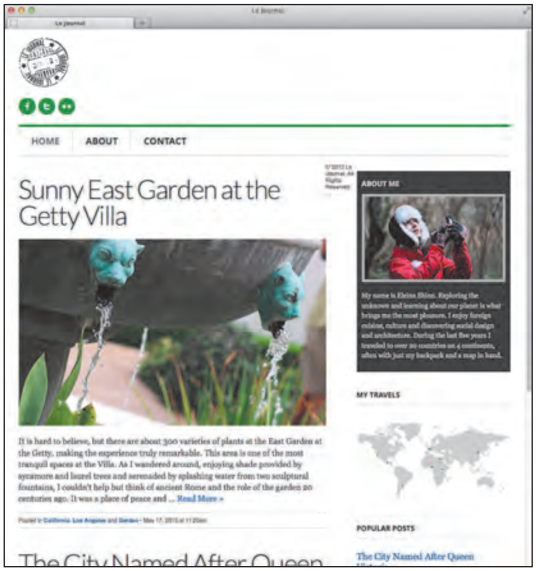


图 11.11.5　瞧，一个两栏布局形成了！不过，两栏之间的东西是什么呢？让我们仔细看看（如图 11.11.6 所示）……

让文本环绕元素

(1) 输入 float:。

(2) 输入 left，让元素浮动到左边，其他内容围绕在它右边；

或者输入 right，让元素浮动到右边，其他内容围绕在它左边；

或者输入 none，让元素完全不浮动。（none

是默认值，因此只有在特定情况下需要覆盖某个浮动元素的样式规则时才需要用到它。）

(3) 使用 width 属性显式地设置元素的宽度（参见 11.7 节），以便有空间放置围绕它的内容。



图 11.11.6　这是页脚！它怎么会在那里呢？首先，记住我们的页面是 960 像素宽，main 是 600 像素宽，而附注栏是 300 像素宽。两栏向相反的方向浮动，中间就会留下 60 像素的空隙。其次，如果你看看 HTML，你会发现 main 的代码就在附注栏的前面，而附注栏就在页脚前面。由于两栏都是浮动的，因此页脚的内容就会流动到它们之间，就像图 11.11.3 中的段落文字环绕图像一样。后面将介绍如何让页脚回到下面的位置

提示　记住，你选择的方向是应用于需要浮动的元素的，而不是应用于环绕它的元素的。在使用 float:left 时，页面的其他部分围绕在右边，反之亦然。

提示　float 属性不是继承的。

11.12　控制元素浮动的位置

我们可以控制浮动什么时候停止，不过在介绍这些内容之前，需要进一步理解 float 属性是如何影响页面的。

考虑图 11.12.1 中的 HTML。这是一段博客文章的介绍（不过没有标题）。可以看到，section 元素包含了所有相关的内容。假设该元素的样式与之前的一样，只是添加了一个花哨的背景色，从而突出其高度（如图 11.12.2 所示）。

```
<section class="post">
    <img ... class="post-photo" />

    <div class="post-blurb">
       <p>An hour and a half ...</p>
    </div>

    <footer class="footer">
       <p class="post-footer">Posted ...</p>
    </footer>
</section>
```

图 11.12.1　在这个简单的结构中，博客文章的所有介绍性内容都放在一个 section 中。在贯穿本章的例子中，<h1>The City Named After Queen Victoria</h1> 位于 section 中 img 前的位置

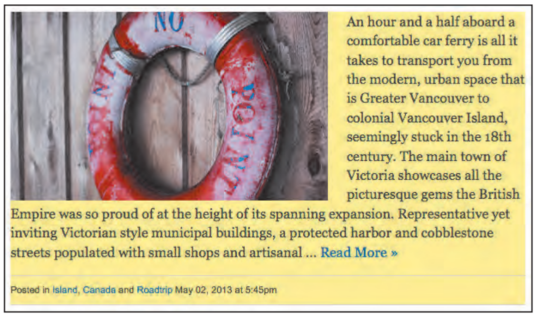


图 11.12.2　图像应用了 float: left;，这样文字就环绕在了它旁边。section 元素（黄色区域）的高度看起来好像受到了图像的影响，但事实并非如此

浮动的元素对文档流的影响与非浮动元素的影响是不同的。如图 11.12.2 所示，图像等浮动元素会让接下来的内容环绕在它周围。不过，它并不会影响父元素或其他祖先元素的高度，因此从这一点来说，它不属于文档流的一部分。

为了演示这一点，我移除了博客摘要中的一部分文字（如图 11.12.3 所示）。你或许

会对此感到惊讶——尽管图像是 section 的子元素，但 section 却比图像更矮一些！只有普通文档流中的内容会影响父元素的高度。在这个例子中，这样的内容包括博客摘要文字和博客文章的页脚，但不包括图像。



图 11.12.3　现在你可以清晰地看到浮动元素是如何影响文档流的了——它完全不影响父元素的高度

页脚为什么会显示在这里呢？它之前的内容变短之后，它移到了图像的旁边。不过，这显然不是我们希望看到的效果。毕竟，你的博客文章的摘要可能很短。如何改变这样的样式呢？

幸好可以使用 clear 属性清除浮动效果（如图 11.12.4 和图 11.12.5 所示）。如果对某个元素使用该属性，该元素和它后面的元素就会显示在浮动元素的下面。

```
.post-footer {
    clear: left;
}
```

图 11.12.4　使用 clear: left;，让内容显示在浮动元素的后面

1. 让浮动元素的父元素"自清除"

回过头来看 11.11 节（参见图 11.11.6），我们还有一个布局问题有待解决。尽管在 HTML 中页面的页脚在 main 和附注栏的后面（如图 11.12.6 所示），但是它并没有显示在 main 和附注栏的后面。

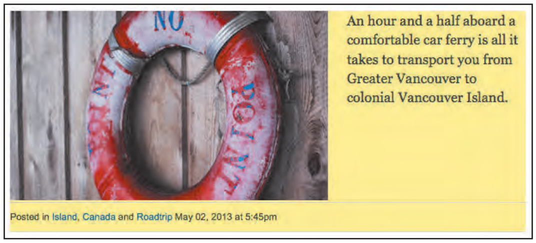


图 11.12.5　现在，博客文章的页脚不会显示在图像的旁边了，无论它前面的文本多短。父元素的高度也会相应增加。可以为页脚添加 margin-top 属性，这样它不会紧贴着图像。一旦清除浮动，浮动效果就没有了，因此不需要为图像后面的所有元素都应用这条规则

```
...
<body>
<div class="page">
    ... [masthead] ...

    <div class="container">
       <main role="main">
          ...
       </main>

       <div class="sidebar">
          ...
       </div>
    </div> <!-- end container -->

    <footer role="contentinfo" ...>
       <p class="legal"><small>&copy;
       2013
       → Le Journal. All Rights
       Reserved.
       → </small></p>
    </footer>
</div>
</body>
</html>
```

图 11.12.6　示例页面的基本结构。带 class="container" 的 div 是 main 和附注栏的父元素。页面的页脚在它们的后面

根据先前定义的样式，main 元素有 float: left;，附注栏有 float: right;。现在你了解了，浮动的元素不会影响其父元素的高度，

这样你就能理解页脚显示在靠近页面顶端位置的原因了。两列内容父元素（作为容器的 div）的高度是 0！不信？看看图 11.12.7。

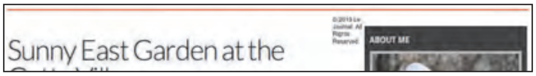


图 11.12.7　如果应用样式 .container { border: 1px solid red; }，你可能认为容器 div 的子元素外围出现一条边框，实际上不是，你会看到边框显示为一条线，位于容器 div 的顶端。这是因为它所有的内容要么是左浮动的，要么是右浮动的，因此它的高度为 0。这就是页面的页脚直接显示在容器下面的原因

可以通过为页脚设置 clear: both; 解决这个问题。这与图 11.12.4 中演示的方法类似，它可以清除浮动，既包括左浮动，也包括右浮动。

这样做可以让页脚显示在浮动的列的下方，但容器 div 的高度仍然是 0。清除浮动的 .post-footer 与浮动元素（图 11.12.1 中的 section，效果如图 11.12.5 所示）位于同一个父元素的里面，但页面的页脚却位于作为容器的父元素（如图 11.12.6 所示）的外面。这意味着它无法影响容器的高度、浮动的元素或不浮动的元素。

大多数时候，这样做没有问题。但如果希望为容器添加一个背景色，由于容器高度为 0，不会显示该背景色。解决的办法是让容器自身具有清除浮动的能力。

有很多方法可以解决这个问题，但最可靠的是使用所谓的 clearfix 方法。要使用这种方法，只需要在样式表中引入 .clearfix 的规则（如图 11.12.8 所示），然后为浮动元素的父元素（该元素为希望清除浮动的元素）添加 clearfix 类（如图 11.12.9 所示），从而可以支撑起父元素，并能够添加一些预期样式（如图 11.12.10 和图 11.12.11 所示）。

```
.clearfix:before,
.clearfix:after {
    content: " ";
    display: table;
}

.clearfix:after {
    clear: both;
}

.clearfix {
    *zoom: 1;
}
```

图 11.12.8　大量网站使用 clearfix 类或类似方法清除浮动。本书不会对所有这些代码进行详细解释，因为有些复杂。不过请将它常备手头，复制到每一个需要用到它的网站的样式表中去

```
…
<div class="container clearfix">
    <main role="main">
        …
    </main>

    <div class="sidebar">
        …
    </div>
</div>
…
```

图 11.12.9　对容器添加 clearfix 类会清除浮动的 main 和附注栏元素，从而让容器的高度等于两列中较高的那一个的高度

```
.container {
    background: url(../img/bg.png)
    → repeat-y 629px 0;
    padding-bottom: 1.9375em;
}

footer[role="contentinfo"] {
    border-top: 1px solid #cacbcb;
}
```

图 11.12.10　这里有一个简单的 5 像素 × 20 像素的背景图像，背景位置为距左侧 629 像素，垂直方向重复，从而为两栏之间提供了一个分隔线。另外，可以为页面的页脚加上一个上边框，将页脚与其他内容分隔开

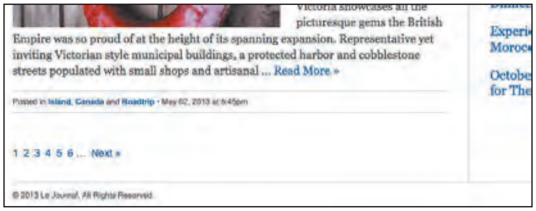


图 11.12.11　页面的页脚的边框与两栏的分隔线是垂直的。为两栏的容器添加的下内边距让分页链接与页脚之间产生了一些空间

2. 控制元素浮动位置的步骤

(1) 输入 clear:（如图 11.12.4 所示）。

(2) 输入 left，阻止元素浮动在该元素的左边；

或输入 right，阻止元素浮动在该元素的右边；

或输入 both，阻止元素浮动在该元素的左右两边；

或输入 none（默认值），允许元素浮动在该元素的左右两边。

(3) 如果要让浮动元素的祖先元素在高度上包含浮动元素，并消除浮动，可以使用 clearfix（参见图 11.12.8 和图 11.12.9）或 overflow 方法替代第 (1) 步和第 (2) 步。（参见补充材料“使用 overflow 创建自清除浮动元素”。）

提 示　应该将 clear 属性添加到不希望环绕浮动对象的元素上（参见图 11.12.4 ~ 图 11.12.6）。因此，如果要让一个元素在右侧没有浮动元素（以及任何靠向右侧的元素）之后才显示，就为它添加 clear: right;（而不是为浮动的元素添加此样式规则）。而 clearfix 和 overflow 方法是应用于浮动元素的父元素或祖先元素的。

提 示　过去的几年，有很多人在不断地优化 clearfix 的代码。图 11.12.8 演示的代码来自 Nicolas Gallagher（见 http://nicolasgallagher.com/micro-clearfix-hack/，不过注意这篇文章讲到的技术比较深），并包含在出色的 HTML5 Boilerplate（www.h5bp.com）项目中（该项目是由 Paul Irish 发起的）。

提 示　浮动元素的 display 属性会变成 display: block;，哪怕将其设置为 display: inline;（无论是浏览器的默认样式还是手动显式设置），该属性值依然为 block。

使用 overflow 创建自清除浮动元素

通常，可以对浮动元素的父元素使用 overflow 属性以替代 clearfix 方法（参见图 11.12.8 和图 11.12.9）。例如，在示例页面中可以使用以下代码：

```
.container {
    overflow: hidden;
}
```

在某些情况下，overflow: hidden; 会将内容截断，对此要多加注意。有时使用 overflow: auto; 也有效，但这样做可能会出现一个滚动条，这显然是我们不希望看到的。有的代码编写人员仅在 overflow 能解决 float 问题的情况下才使用 overflow 方法，在其他情况下则使用 clearfix 方法。有的代码编写人员则始终使用 clearfix 方法。

顺便说一下，可以将 clearfix 或 overflow 应用到浮动元素的任何一个非父元素的祖先元素。这样做不会让父元素变高，但祖先元素的高度会包含浮动元素。

关于 overflow 属性的更多信息，参见 11.16 节。

11.13　对元素进行相对定位

每个元素在页面的文档流中都有一个自然位置（参见图 11.13.1 和图 11.13.2）。相对于这个原始位置对元素进行移动就称为相对定位（如图 11.13.3 和图 11.13.4 所示）。周围的元素完全不受此影响(如图 11.13.4 所示)。

```
…
<h1>Relative Positioning</h1>

<p>When you position an element relatively,
→ you <span class="example">position it
→ </span> relative to its normal location.
→ </p>
…
```

图 11.13.1　一会儿要定位的 span 文本

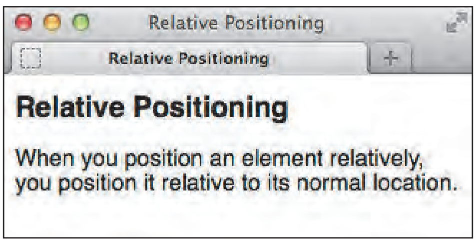


图 11.13.2　这是一个普通段落，这里的呈现没什么奇怪的

```
.example {
     position: relative;
     top: 35px;
     left: 100px;
}
```

图 11.13.3　记住，既要指明采用相对位置，还要给出偏移量。可以使用正数，也可以使用负数。为简化示例，这里使用的是像素，使用 em 会使偏移量的大小与文本字体大小成比例，即便文字变小或者变大都一样。由于 1em 等于元素的字体大小，默认情况下，文本通常为 16 像素，因此这个例子中的 top: -3em;，会使文本向下移动 48 像素

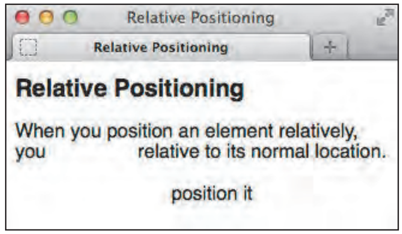


图 11.13.4　相较于先前的位置（现在显示为一块空白），文本与上边相距 35 像素，左边相距 100 像素。如果设置为 top: -55px;，它的位置就会在 h1 文本的上面

偏移自然流中元素的步骤

(1) 输入 position: relative;。

(2) 输入 top、right、bottom 或 left。再输入 :*d*，这里的 *d* 是希望元素从它的自然位置偏移的距离，可以表示为绝对值（如 10px）、相对值（如 2em）或百分数。值可负可正。需要添加其他偏移量，可重复这一步。

提示　其他元素不会受到偏移的影响，仍然按照这个元素原来的盒子进行排列。设置了相对定位的内容可能与其他内容重叠，这取决于 top、right、bottom 和 left 的值。

提示　使用相对定位、绝对定位或固定定位时，对于相互重叠的元素，可以用 z-index 属性指定它们的叠放次序。详细信息参见 11.15 节。

提示　对元素设置 position: static，可以覆盖 position: relative 设置。static 是元素的默认值，这就是元素出现在常规文档流中的原因。有关示例参见 11.15 节。

提示　定位不是继承的。

11.14　对元素进行绝对定位

正如前面提到的，网页中的元素通常按照它们在 HTML 源代码中出现的次序进行排列（如图 11.14.1 和图 11.14.2 所示），除非你使用 CSS 规则改变前面的样式。也就是说，如果 img 元素出现在 p 元素之前，图像就出现在段落的前面。

通过对元素进行绝对定位，即指定它们相对于 body （参见图 11.14.3 和图 11.14.4）或最近的已定位祖先元素（参见图 11.14.5 和图 11.14.6）的精确位置，可以让元素脱离正常的文档流。

这与相对定位不同，绝对定位的元素不会在原先的位置留下空白。这与让元素浮动也不同。对于绝对定位的元素，其他元素不会环绕在它的周围。事实上，其他内容不知道它的存在，它也不知道其他内容的存在。

```
…
<header class="masthead" role="banner">
    <p class="logo"><a href="/"><img … />
    → </a></p>

    <ul class="social-sites">
        …
    </ul>

    <nav role="navigation">
        <ul class="nav-main">
            …
        </ul>
    </nav>
</header>
…
```

图 11.14.1　页面的报头（或为页眉）依次包含了网站的标识、社交网站图标和主导航。它们的默认显示方式如图 11.14.2 所示。接下来你将看到如何通过两个突出显示的类，让社交网站图标在报头中进行绝对定位

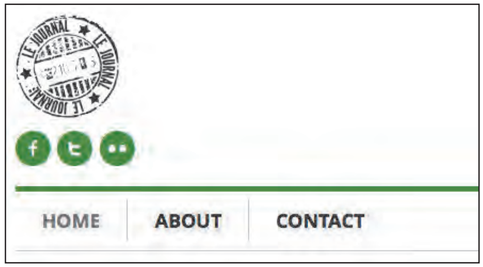


图 11.14.2　在默认文档流里，社交网站图标的列表位于网站标识和导航之间。我们希望它显示在包含它的报头里自上而下一半的位置，并让它显示在报头的右侧

```
.masthead .social-sites {
    position: absolute;
    top: 41px;
    right: 0;
}
```

图 11.14.3　通过对图标列表进行绝对定位，我们让它完全脱离了文档流。仅用这段代码还不能实现目的，因为在没有另外进行指定的情况下，设置 position: absolute 的元素是相对于 body 进行定位的，如图 11.14.4 所示

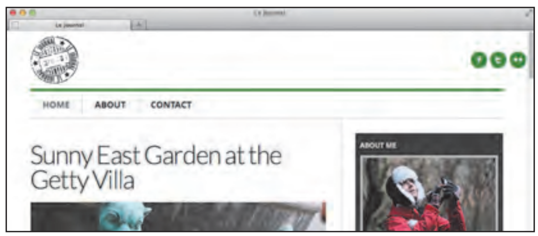


图 11.14.4　图标列表显示在距离 body 上边缘 41 像素、右边缘 0 像素的位置。很明显这种处理方法不好，因为用户将浏览器窗口放大得越宽，图标列表就跟页面中的其他内容距离越远

对元素进行绝对定位的步骤

(1) 输入 position: absolute;。

(2) 根据需要，输入 top、right、bottom 或 left。

再输入 :*d*，这里的 *d* 是希望元素相对于其祖先元素偏移的距离（如 10px 或 2em）或

相对于其祖先的百分数（有关解释参见第二条提示）。

```
.masthead {
    position: relative;
}

.masthead .social-sites {
    position: absolute;
    top: 41px;
    right: 0;
}
```

图 11.14.5　对包含图标列表（参见图 11.14.2）的页眉设置 position: relative，从而让这些图标可以相对页眉（而不是 body 元素）进行绝对定位。这样就可以让图标显示在我们希望它显示的位置，如图 11.14.6 所示

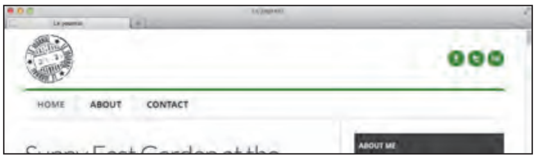


图 11.14.6　完美了

(3) 根据需要，对其他方向重复第 (2) 步。

(4) 根据需要，对希望成为绝对定位参照体的**祖先**元素添加 position: relative;（如图 11.14.5 和图 11.14.6 所示）。如果跳过这一步（如图 11.14.3 所示），将对元素相对于 body 计算偏移量（如图 11.14.4 所示）。

提示　由于绝对定位的元素脱离了文档流，因此它们可能会相互重叠，或与其他元素重叠（这不一定是坏事儿）。

提示　如果不为绝对定位的元素指定偏移量，这个元素将显示在它的自然位置上，但不会影响后续元素在文档流中的位置。

提示　还有一种定位类型称为固定定位。当访问者滚动浏览器窗口时，页面内容通常随之上下移动。如果对元素设置 position: fixed;，它就会固定在浏览器窗口中。当访问者上下滚动浏览器窗口时，该元素不会随之移动，页面的其余部分仍照常滚动。这跟固定背景图像的工作原理类似（参见 10.10 节）。固定定位在很多移动浏览器中效果不佳，因此如果你希望自己的网页能很好地适应移动设备，最好不要使用固定定位。

提示　使用相对定位、绝对定位或固定定位时，可以使用 z-index 属性指定相互重叠的元素的叠放次序。详细说明参见 11.15 节。

提示　对元素设置 position: static 将覆盖 position: absolute; 的设置。static 是元素定位的默认值，这也是元素为什么出现在常规文档流中的原因。有关实例参见 11.15 节。

提示　定位不是继承的。

11.15　在栈中定位元素

当你开始使用相对定位、绝对定位和固定定位以后，很可能发现元素相互重叠的情况，这时可以选择哪些元素应该出现在顶层（参见图 11.15.1、图 11.15.2 和图 11.15.3）。

```
<div class="box1"><p>Box 1</p></div>
<div class="box2"><p>Box 2</p></div>
<div class="box3"><p>Box 3</p></div>
<div class="box4"><p>Box 4</p></div>
```

图 11.15.1　HTML 中 div 的次序

```
div {
    border: 1px solid #666;
    height: 125px;
    position: absolute;
    width: 200px;
}

.box1 {
    background: pink;
    left: 110px;
    top: 50px;
    z-index: 120;
}

.box2 {
    background: yellow;
    left: 0;
    top: 130px;
    z-index: 530;
}

.box3 {
    background: #ccc;
    position: static;
    /* 静态的，没有任何效果 */
    z-index: 1000;
}

.box4 {
    background: orange;
    left: 285px;
    top: 65px;
    z-index: 3;
}
```

图 11.15.2 对于定位元素，z-index 最高的数显示在最上面（如图 11.15.3 所示），不管该元素在 HTML 中出现的次序（如图 11.15.1 所示）。第一条规则为所有四个 div 设置了 position: absolute;，而定义 .box3 时又覆盖了这一规则，让 .box3 回到默认的 static。因此，尽管它的 z-index 值最高，但这没有任何效果，它总是位于最底层

在栈中定位元素

输入 z-index: *n*，其中的 *n* 是表示元素在定位过的对象堆中的层级数字。

提示 z-index 的值越大，元素在堆中就越高（参见图 11.15.2 和图 11.15.3）。

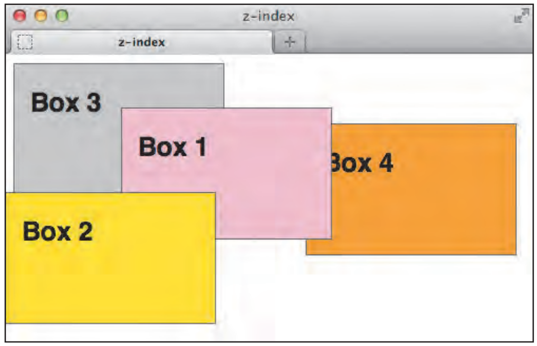


图 11.15.3 定位的盒子按照 z-index 由高到低的次序进行叠放。第三个盒子位于最底层，因为它处于常规的文档流

提示 z-index 属性仅对定位过的元素（即设为绝对定位、相对定位或固定定位的元素）有效。图 11.15.1 仅包含绝对定位的元素，但实际上可以对绝对定位、相对定位和固定定位的元素混合使用 z-index，z-index 会将它们作为整体进行安排，而不是分别安排。

提示 z-index 属性不是继承的。

11.16 处理溢出

元素并不总是包含在它们自己的盒子里。这可能是因为盒子不够大，例如，图像比它的容器更宽就会溢出；也可能是因为使用负的外边距或绝对定位让内容处于盒子的外面；还有可能你对元素设置了显式高度，它的内容太高而无法装入盒子内部。无论是哪种情况，都可以使用 overflow 属性控制元素在盒子外面的部分，参见图 11.16.1 和图 11.16.2。

11

```
div {
    background: #e0f7ac;
    border: 1px solid #666;
    height: 88px;
    width: 300px;
}
.example-hidden { /* div 2 */
    overflow: hidden;
}
.example-scroll { /* div 3 */
    overflow: scroll;
}
.example-auto { /* div 4 */
    overflow: auto;
}
```

图 11.16.1　这里对 div 设置了高度，不过通常最好不要对元素设置高度。这里为 div 添加了背景色和边框，很容易看出不设置高度的原因。三个类演示了控制内容溢出的三种方式

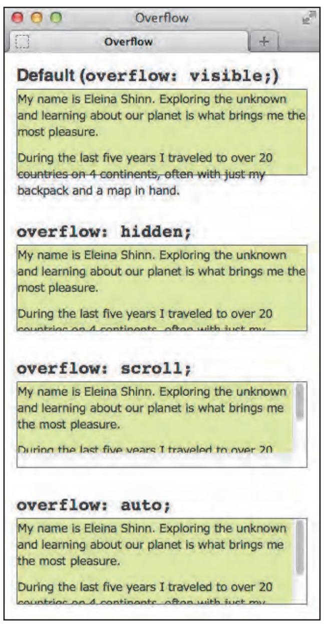


图 11.16.2　每个带边框的绿色区域都是一个包含两个段落的 div。浏览器会照顾对元素指定的高度。如果内容太高，就会穿过 div，透到外面来（就像第一个 div 那样），除非使用其他 overflow 设置，覆盖默认的样式

决定浏览器如何处理溢出的步骤

(1) 输入 overflow:。

(2) 输入 visible，让元素盒子中的所有内容可见，这是默认项；

或者输入 hidden，隐藏元素盒子容纳不了的内容；

或者输入 scroll，无论元素是否需要，都在元素上添加滚动条；

或者输入 auto，让滚动条仅在访问者访问溢出内容时出现。

提 示　如果图 11.16.1 中的代码使用 min-height: 88px; 代替 height 88px;，图 11.16.2 中的 div 就会随着内容的增加而变大。如果内容的高度比 88 像素矮，div 仍为 88 像素高，因为 min-height 设置的是最小高度。

提 示　overflow 属性还可以很方便地清除 float，参见 11.12 节的补充材料。

提 示　overflow 属性不是继承的。

11.17　垂直对齐元素

可以使用除默认对齐方式以外的多种方式对齐元素，让它们在页面上显得较为整齐（如图 11.17.1 ~ 图 11.17.4 所示）。

```
…
<form action="results.php" method="get"
→ role="search">
    <label for="search">Search</label>
    <input type="search" id="search"
    → name="search" />
    <input type="image" src="img/btn-go.png"
    → alt="Submit search" />
</form>
…
```

图 11.17.1　这是一个简单的表单，只有一个文本标签、一个文本输入框和一个图像提交按钮。关于表单的更多信息，参见第 16 章

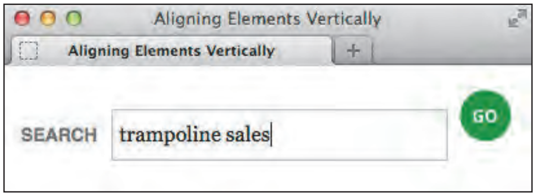


图 11.17.2　默认情况下，行内内容在竖直方向上与文本的基线对齐。这里在表单输入框里填入了一些文本，用以演示这一情况——标签文本（SEARCH）、输入框中的文字与图像的底端都与基线对齐

```
input[type="image"] {
    vertical-align: bottom;
}
```

图 11.17.3　每一行内容都有一个看不见的框（线框），它代表了行的高度。在这个例子中，文本输入字段就能指示框的下边缘，因为它是这一行最低的部分（如图 11.17.2 所示），图像会对齐线框的底部（如图 11.17.4 所示）

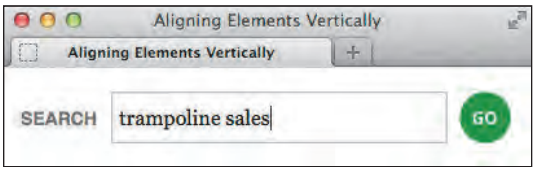


图 11.17.4　搞定！

使元素垂直对齐的步骤

(1) 输入 `vertical-align:`。

(2) 输入 `baseline`，使元素的基准线对齐父元素的基准线；

或者输入 `middle`，使元素位于父元素中央；

或者输入 `sub`，使元素成为父元素的下标；

或者输入 `super`，使元素成为父元素的上标；

或者输入 `text-top`，使元素的顶部对齐父元素的顶部；

或者输入 `text-bottom`，使元素的底部对齐父元素的底部；

或者输入 `top`，使元素的顶部对齐当前行里最高元素的顶部；

或者输入 `bottom`，使元素的底部对齐当前行里最低元素的底部；

或者输入元素行高的百分比，可以是正数，也可以是负数；

或者输入某个值（正负均可，单位为像素或 em）分别按照特定的值向上或者向下移动元素。

更多关于 vertical-align 的说明

可以使用 `vertical-align` 设置表格单元格中内容的对齐方式。通常，默认的样式是中间对齐，而不像表格以外的内容那样与基线对齐。我们将在第 18 章讲解表格。除了表格以外，`vertical-align` 属性仅适用于行内元素，不能应用于块级元素。更多信息参见 Chris Coyier 的解释（http://css-tricks.com/what-is-vertical-align/）。

11.18　修改鼠标指针

一般情况下，浏览器负责控制鼠标指针的形状。大多数时候使用箭头形状，指向链接时使用手指形状（参见图 11.18.1），等等。使用 CSS 可以修改指针的形状（如图 11.18.2 和图 11.18.3 所示）。



图 11.18.1　指向 Home 链接时，指针变成手指形状，同时链接高亮显示。对于其他链接也是这样

```
.nav-main .current-page,
.nav-main .current-page:hover {
    color: #747474;
    cursor: default;
}
```

图 11.18.2　当访问者位于主页时，我为 HOME 链接添加了 class="current-page"。这样就修改了链接文本的颜色、鼠标停留状态的颜色和指针形状，让它看起来不像是链接（另外，在这个例子中，也可以在导航中去掉 HOME 链接周围的 a 元素）

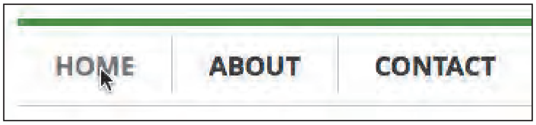


图 11.18.3　尽管它仍然是真正的链接，但它看起来已不像链接。访问者正位于这个链接指向的页面，因此这样做是有意义的

修改指针形状的步骤

(1) 输入 cursor:。

(2) 输入 pointer 表示停留在链接上时通常显示的指针形状（）或 default 表示箭头（），或者输入 crosshair（＋）、move（）、wait（）、help（）、text（）或 progress（）；

或者输入 auto 显示特定情形下通常使用的指针形状；

或者输入 *x*-resize 显示双向箭头，这里的 *x* 是其中一个箭头需要指向的方向，可以是 n（北）、nw（西北）、e（东），等等。例如，e-resize 指针显示成。

> **提示**　不同浏览器、不同系统的指针形状可能有细微的差异。

第 12 章

构建响应式网站

本章内容

- ❑ 响应式 Web 设计：概述
- ❑ 创建可伸缩图像
- ❑ 创建弹性布局网格
- ❑ 理解和实现媒体查询
- ❑ 汇总
- ❑ 兼容旧版 IE

最后一分钟决定要不要去看电影；就安道尔的官方语言跟人打赌；查找已经迟到 15 分钟的会议所在公司的电话号码；查看去这家公司的地图（因为迟到的原因是找不到这家公司）……

我们想要及时地获取信息，而随着各种形状和尺寸的移动设备的剧增，万维网可以出现在你的口袋里、钱包里、背包里，就像它在书桌、餐桌和卧室一样方便。

现在，网站的建设者需要让访问者能够通过移动电话、智能手机、平板电脑、笔记本电脑、台式机、游戏机、电视机以及未来任何可以上网的设备获取信息。响应式 Web 设计就是为此诞生的。

在本章中，你将了解到如何构建在各种设备上都能正常工作的网站，它能根据设备的功能和特征对布局进行调整。

12.1　响应式 Web 设计：概述

以前，为了满足移动用户的需求，需要额外建立一个专门为移动设备设计的网站。不久之前这种做法还比较流行，但是目前已经越来越少见了。不过，现在也还有一些公司在这样做。有的还为平板电脑建立了第三个网站。

不存在可以适应所有情形的某种正确方法。每周都有新的设备投放到市场上，而且毋庸置疑的是，各家公司都在设计新的设备类型。那么，构建和维护多个网站现实吗？甚至说，有必要吗？我们无法预知未来的情形。

幸好，我们有办法构建一个既可以运行在现在的设备，也可以运行在未来设备上的网站。更棒的是，可以让它在较小的屏幕、较大的屏幕以及介于两者之间的屏幕上显示不同的效果（如图 12.1.1 ～图 12.1.3 所示）。



图 12.1.1　不管你信不信，这里以及图 12.1.2 和图 12.1.3 显示的 Food Sense 主页（www.foodsense.is）都来自同一个网站，而不是各自具有单独 URL 的不同网站。这个网站采用了响应式 Web 设计方法，因此它的布局可以根据不同的查看环境进行改变。iPhone（如本图所示）及其他屏幕大小相似的设备会根据特定的 CSS 规则显示布局。对于更大的浏览器窗口（参见图 12.1.2 和图 12.1.3），则有另外一些 CSS 规则控制相应的布局

响应式页面的组成

Ethan Marcotte 为我们提供了如何实现上述效果的蓝图，并将相应的方法命名为响应式 Web 设计（responsive Web design）。关于响应式设计的历史参见 11.1 节。他的方法植根于以下三点。

- 灵活的图像和媒体。图像和媒体资源的尺寸是用百分数定义的，从而可以根据环境进行缩放。
- 灵活的、基于网格的布局，也就是流式布局。对于响应式网站，所有的 `width` 属性都用百分数设定，因此所有的布局成分都是相对的。其他水平属性通常也会使用相对单位（em、百分数和 rem 等）。
- 媒体查询。使用这项技术，可以根据媒体特征（如浏览器可视页面区域的宽度）对设计进行调整。



图 12.1.2　这是使用 iPad 及其他屏幕大小相似的设备查看 Food Sense 主页的样子。由于浏览器有更多空间显示内容，因此这套样式的 CSS 对标识和导航都进行了调整

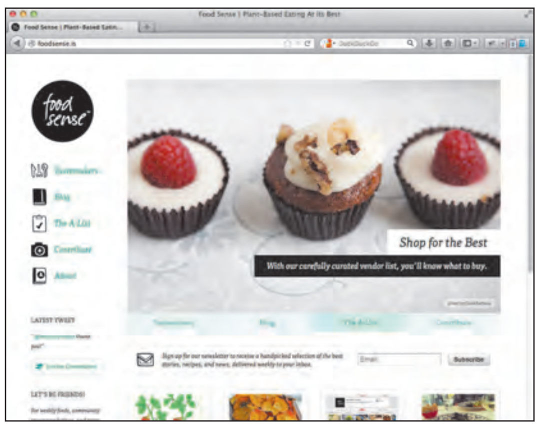


图 12.1.3　这是 Food Sense 主页显示在桌面浏览器上的样子，也是最宽的显示效果。该站还有另外两个布局没有展示出来。你可以在计算机上访问 www.foodsense.is，再拖动浏览器的一角让浏览器窗口变窄或变宽并查看效果

我们将在接下来的几章逐一介绍这些主题。随后，你将一步步学习如何构建第 11 章示例的响应式版本。

提示　John Allsopp 的文章“Web 设计之道”（“A Dao of Web Design”，http://alistapart.com/article/dao）写于 2000 年，该文讨论了设计和构建灵活的网站的方法。这篇文章是响应式 Web 设计的先驱，Marcotte 以及很多其他的作者都引用过这篇文章，且认为它影响巨大。这篇文章很值得一读。

提示　Jeremy Keith 在题为“One Web”的演讲中归纳了“一个网站适应所有设备”的方法（www.vimeo.com/27484362/）。讲稿全文参见 www.adactio.com/articles/4938/。

12.2　创建可伸缩图像

正如你在第 5 章学到的，默认情况下，图像显示的尺寸是 HTML 中指定的 width 和 height 属性值（参见图 12.2.1）。如果不指定这些属性值，图像就会自动按照其原始尺寸显示。此外，你还可以通过 CSS 以像素为单位设置 width 和 height。

显然，当屏幕宽度有限的时候，按原始尺寸显示图像就不一定合适了（如图 12.2.2 所示）。使用可伸缩图像技术（如图 12.2.3 和图 12.2.4 所示），就可以让图像在可用空间内缩放，但不会超过其本来的宽度（如图 12.2.5 和图 12.2.6 所示）。可用空间是由包含图像的元素决定的（参见最后一条提示）。在图 12.2.5 中，可用空间是由 body 元素决定的，在图 12.2.6 中则是 main 元素。

```
...
    <meta name="viewport"
    → content="width=device-width,
    → initial-scale=1" />
</head>
<body>
<img src="img/gettyvilla.jpg" width="600"
→ height="365" alt="" class="post-photo-full"
→ />

<img src="img/victoria.jpg" width="370"
height="220" alt="" class="post-photo" />
</body>
</html>
```

图 12.2.1　一些典型的指定了 width 和 height 的 img 标签（参见图 12.2.2）。这些代码出现在第 11 章的代码中（注意，这里省略了 alt 中的文本，是为了让代码更简短）



图 12.2.2　使用固定的 width 和 height 值时，即便可视区域变小，图像也依然显示其原始尺寸。桌面浏览器会显示滚动条（左），而在手机上（右），图像占据了整个屏幕

```
...
<img src="img/gettyvilla.jpg" alt=""
→ class="post-photo-full" />

<img src="img/victoria.jpg" alt=""
→ class="post-photo" />
...
```

图 12.2.3　只有 HTML 中不包含 width 和 height 属性，图像才有可能变成可伸缩的图像。你还可以为图像添加用于添加样式的类

```
.post-photo,
.post-photo-full {
    max-width: 100%;
}
```

图 12.2.4　让图像变成可伸缩图像的一小段魔法 CSS

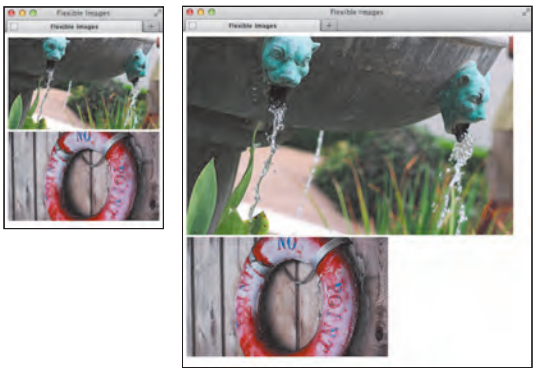


图 12.2.5　可伸缩图像可以根据包含它们的元素（在这个例子中为 body）的尺寸按比例缩放。它们不会比其本来的宽度更宽，就像下面的图像所演示的那样。在这个例子中，上面的图像的最大宽度为 600 像素，下面的为 370 像素

制作可伸缩图像的步骤

(1) 对任何想做成可伸缩图像的图像，在 HTML 的 img 标签中省略 width 和 height 属性（如图 12.2.3 所示）。

(2) 在样式表中，为每个想做成可伸缩图像的图像应用 max-width: 100%（如图 12.2.4 所示）。



图 12.2.6　我们在本章将要创建的响应式页面在 320 像素宽的区域（左）和 480 像素宽的区域（右）中显示的样子。图像会缩放以适应容器！

提 示　图像缩放的可用空间是由其父元素建立的内容区域。如果父元素有水平方向上的内边距，可用空间就会相应减小。关于内容区域的尺寸，参见 11.7 节。

提 示　一定要使用 max-width: 100% 而不是 width: 100%。它们都能让图像在容器内缩放，不过，width: 100% 会让图像尽可能地填充容器，如果容器的宽度比图像宽，图像就会放大到超过其本来尺寸，有可能会显得较为难看。

提 示　图标字体和 SVG 可以创建无损缩放的图形。这两个主题分别参见 13.2 节和 5.1 节。

提 示　不要忘了对图像进行优化，让文件尺寸尽可能小，参见 5.4 节。

提 示　这项技术对已经为 Retina 显示屏扩大到双倍大小的图像也适用。当然，双倍分辨率的图像的文件大小也会大很多。关于如何创建为 Retina 显示屏准备的图像，参见 5.8 节（其中还包含了一条帮助减小文件尺寸的提示）。

提示　可以使用 background-size 属性对背景图像进行缩放（对 IE8 无效）。更多信息参见 www.css3.info/preview/background-size/。

提示　还可以使用 video，embed，object { max-width: 100%; } 让 HTML5 视频及其他媒体变成可伸缩的（同样也不要在 HTML 中为它们指定 width 和 height）。对于视频无法按照预期进行缩放的情况，参见 Jonathan Nicol 的文章（http://f6design.com/journal/2011/10/18/responsive-elements-that-retain-their-aspect-ratio/）。

响应式设计与图像之谜

可伸缩图像很棒，对吧？“不过，等等，”你可能会说，“加载一个比在页面上需要显示的图像更大的图像，不是很不好吗？尤其是在连接速度较慢的移动设备上？”

你说得没错，这也是人们试着处理响应式 Web 设计时遇到的最大的问题。

理想情况下，我们需要的是在特定设备上刚好符合尺寸要求的图像。移动设备上用小图像，平板电脑上用大一些的，等等。这能让文件大小变小，同时减小设备用于显示图像的内存消耗（这对手机来说尤其重要）。特别是，这样做能加快页面加载速度。这对那些需要为移动数据计划付费的用户来说也相当友好（移动数据计划限制了用户每个月可以下载的数据大小）。

此外，有时你可能希望在移动设备上显示不一样的图像，而不仅仅是缩小的图像。一个常见的例子是人的照片。小的屏幕只显示人脸的部分，而大的屏幕则显示全身照。

很多人致力于解决这一问题，但到本书写作之际，还没有一种简单的解决办法。

有两个建议为 HTML 增加功能的提案。一个建议是为 img 增加一个名为 srcset 的属性，另一个建议是增加一个新的名为 picture 的元素（参见 http://www.w3.org/community/respimg/）。它们解决的是相似的问题，但并非同一个问题。因此这两个提议有同时变为现实的可能。敬请期待！

同时，在页面中引入图像的时候也要聪明一些。我们的示例页面中最大的图像为 600 像素宽（如图 12.2.6 所示），比智能手机需要的尺寸（大多为 320 像素宽）要大一些，因此访问者有可能察觉到需要较长时间才能加载。幸好页面上只需要显示这一个图像。至于为 Retina 显示屏准备图像，参见倒数第三条提示。

12.3　创建弹性布局网格

拥有定宽容器的网页显得有些死板。前一章创建的例子就是这样（参见图 12.3.1 ~ 图 12.3.3）。如果桌面浏览器的宽度小于页面宽度，就会出现横向滚动条（如图 12.3.4 所示）。

```
...
<body>
<div class="page">
     <header class="masthead" role="banner">
        ...
     </header>

     <div class="container">
        <main role="main">
           ...
        </main>

        <div class="sidebar">
           ...
        </div>
     </div>

     <footer role="contentinfo"
     → class="footer">
        ...
     </footer>
</div>
</body>
</html>
```

图 12.3.1　在第 11 章创建的页面中，所有的内容都处于一个 page 类的 div 里。其中的基本组成包括报头（页眉）、包含 main 和附注栏区域的 div（带有 container 类），以及页面的页脚

```
.page {
     width: 960px;
}

main {
     width: 600px;
}

.sidebar {
     width: 300px;
}
```

图 12.3.2　这些是为第 11 章创建的示例页面（参见图 12.3.1）设置的固定宽度。我们没有为页眉、页脚以及包含 main 和附注栏的 container div 设置宽度，而是让它们使用浏览器的默认值，即 width: auto;，因此它们会跟其父元素一样宽。它们的父元素是 page div（参见图 12.3.1），因此它们也是 960 像素宽（如图 12.3.3 所示）

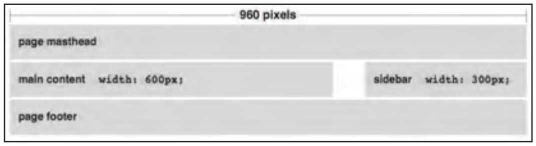


图 12.3.3　最起码，页面要包含四个主要内容区域。.page div 和 .container div 没有显示在这里，是因为它们本身并不包含任何内容。.container div、报头和页脚没有指定宽度（参见图 12.3.2），它们的宽度自动与 .page div 相同。（注意：图 12.3.2 和图 12.3.6 中的 CSS 不会让 main 和附注栏左右并排显示。要知道如何实现这种效果，参见 11.11 节）

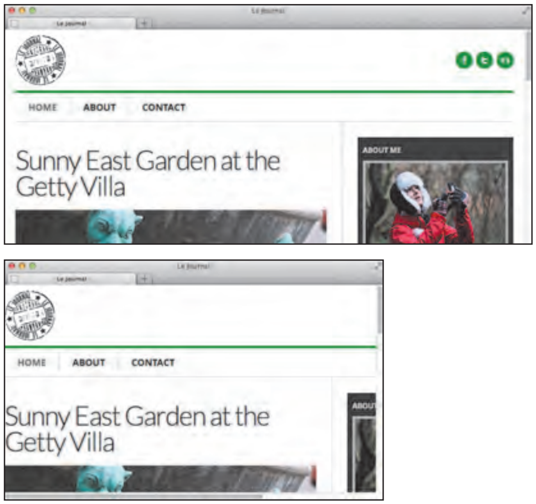


图 12.3.4　即便让浏览器窗口变窄，宽度仍然保持不变

移动浏览器会显示完整的宽度，但页面会显得特别小，这并不太友好（如图 12.3.5 所示）。

这对响应式页面来说并不合适。我们希望页面能进行缩放，并正好适应浏览器视觉区域大小，就像可伸缩图像一样。流式布局（又称弹性布局）便可以做到这一点。

一旦掌握了弹性布局方法，创建弹性布局就很容易了。创建弹性布局需要使用百分数宽度，并将它们应用于页面里的主要区域

（参见图 12.3.1 和图 12.3.6）。这需要一些数学知识，不过不必担心，它不会比你小学数学学到的东西复杂。如果你怀疑这一点，可以在手头准备一个计算器！



图 12.3.5　像 iPhone 这样的手机会缩小页面（如果你希望的话），但访问者需要放大才能阅读里面的内容

```
.page {
    max-width: 960px;
}

main {
    /* 要求宽度/包含块宽度使用600px/960px */
    width: 62.5%;
}

.sidebar {
    width: 31.25%; /* 300px/960px */
}
```

图 12.3.6　这是与图 12.3.2 中定宽布局对等的流式布局样式。为 `.page div` 设置的死板的 `width: 960px;` 被替换为 `max-width: 960px;`。这让该元素可以进行缩放，但不会超过 960 像素。我用先前提供的公式来确定 `main` 和 `.sidebar` 的百分数宽度

那么我们如何知道需要使用多大的百分数呢？实际上，元素的百分数宽度基于其父元素（即包含该元素的容器）提供的可用空间。这些说法似乎有些熟悉？这是因为创建弹性布局的原理与创建可伸缩图像的原理是相同的。可以使用下面的公式计算需要使用的百分数：

> 要指定的宽度（以像素为单位）/ 容器宽度（以像素为单位）= 值

这与 Ethan Marcotte 的公式（目标 / 环境 = 值）是一致的，只是解释得更为明了。如果创建了定宽布局，那么就将它们带到公式里算一下。

1. 一个真实的例子

让我们考虑一个简单的页面布局（将在本章后面构建该页面）。我将演示如何算出 `main` 的 `width` 为 62.5%（参见图 12.3.6）。使用公式，不难得出：

> 要指定的宽度（600px）/ 容器宽度（960px）= 值（.625）

然后将值转化为百分数（将小数点向右移动两位），得到 62.5%。将这个数值作为 `width` 的值（参见图 12.3.6）。完成！

如果你对这一数字还是感到困惑，我会详细解释。布局最宽为 960 像素，这是由 `.page` 的样式规则确定的（参见图 12.3.6）。如果不对 `main` 的父元素 `.container div` 指定宽度（参见图 12.3.1），其宽度就自动为 960 像素。（`.container div` 的父元素是 `.page div`。）

如果我希望 `main` 的宽度最多占据 960 像素中的 600 像素。使用先前给出的公式，600/960=0.625。使用百分数，则为 62.5%。对 `.sidebar` 也可以使用同样的方法，只不过我希望其宽度不超过 300 像素。因此使用 300/960 计算，得 0.3125，即 31.25%（参见图 12.3.6）。

结合使用可伸缩图像和弹性布局，便可

以让整个页面变得可以缩放（如图 12.3.7 和图 12.3.8 所示）。

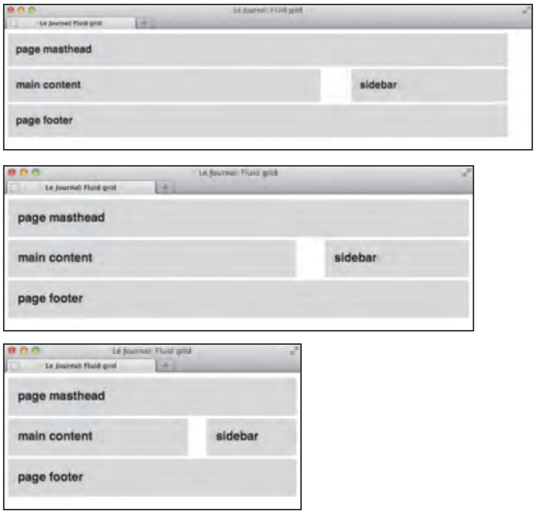


图 12.3.7　无论页面的视觉区域有多宽，两栏始终保持各自所占的比例。如果可视区域宽度超过 960 像素，页面就停止变宽，如最上面的图所示

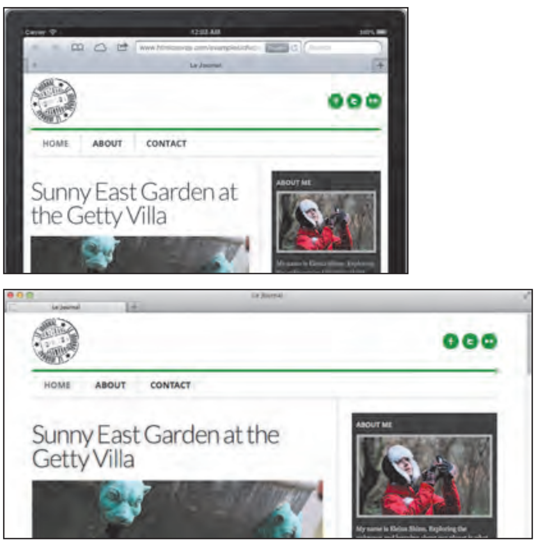


图 12.3.8　应用到本章后面将要建立的响应式页面，图 12.3.6 中给出的样式会让布局可以缩放，不过永远不会超过 960 像素，无论浏览器窗口有多宽，如下面的图所示

2. 创建弹性布局的步骤

对于需要某个宽度实现预期布局的元素，设置 `width: percentage;`，其中 percentage 表示你希望元素在水平方向上占据容器空间的比例（参见图 12.3.6）。通常说来，不必设置 `width: 100%;`，因为默认设置为 `display: block;` 的元素（如 p 以及其他很多元素）或手动设置为 `display: block;` 的元素在默认情况下会占据整个可用空间。

作为可选的一步，对包含整个页面内容的元素设置 `max-width: value;`，其中 value 表示你希望页面最多可增长到的最大宽度（参见图 12.3.6、图 12.3.7 和图 12.3.8）。通常，value 以像素为单位，不过也可以使用百分数、em 值或其他单位的值。

提 示　如果父元素有水平方向上的内边距，它为子元素建立的容器就会变小。要回顾这段内容，请参见 11.7 节。

提 示　还可以对元素设置基于百分数的 margin 和 padding 值。在示例页面中，我对这些属性使用的是 em 值，这是一种常见的做法。内边距和外边距的 em 值是相对于元素的 font-size 的，而基于百分数的值则是相对于包含元素的容器的。

提 示　对于设置了 `body { font-size: 100%; }` 的页面，对 font-size、margin、padding 和 max-width 使用 em 值还有一个好处——如果用户更改了浏览器默认字体大小，那么页面也会跟着变大或变小。例如，在 Firefox 中，可以在 Preferences 中的 Content 标签页中更改默认字体大小。不妨打开 www.htmlcssvqs.com/8ed/examples/12/finished-page.html 试试这一特性。

提示 在使用公式计算元素的百分数宽度时，不要忘了容纳它的容器来自它最直接的祖先（即上下文）。

提示 将box-sizing属性设置为border-box，就可以很方便地对拥有水平方向内边距（使用em或其他的单位）的元素定义宽度，而不必进行复杂的数学计算来找出百分数的值。这对响应式页面来说很方便。更多相关信息参见11.5节。

设置相对的max-width

包含整个页面的.page div（参见图12.3.1）设置了以像素为单位的max-width（960），参见图12.3.6。这个值能否使用灵活的相对单位呢？其实，设置.page { max-width: 60em; }就可以了。下面是具体的解释：

em宽度是基于元素字体大小的。例如，如果其字体大小等价于14像素，则width: 10em;会将宽度设置为140像素。

.page div没有指定font-size，因此它从父元素（body元素）继承。我们知道，body的默认字体大小通常等于16像素。因此，如果希望将.page的最大宽度设为960像素，960/16=60em。

设置.page { max-width: 60em; }与使用960px是相似的，但它们有一个显著的不同：前者会根据浏览器默认字体大小进行缩放。第三条提示里讲了如何看到这一效果。

12.4 理解和实现媒体查询

我们在8.6节中学习过，可以使用两种方式针对特定的媒体类型定位CSS。（还有第三种方式，即使用@import规则，我们不讨论这种方法，因为它会影响性能。）回顾一下，第一种方式是使用link元素的media属性，位于head内，参见图12.4.1。第二种方式是在样式表中使用@media规则，参见图12.4.2。

```
…
<head>
    …
    <link rel="stylesheet" href=
    ➝"your-styles.css" media="screen" />
</head>
…
```

图12.4.1 在屏幕上查看页面时会应用your-styles.css中的样式

```
/* 屏幕和打印共用的样式 */
…

/* 只用于打印的样式 */
@media print {
    header[role="banner"] nav,
    .ad {
        display: none;
    }
}
```

图12.4.2 通过@media print规则可以创建专门为打印浏览器里的页面定义的样式。也可以将它们与为其他媒体定义的样式放在一起

媒体查询增强了媒体类型方法，允许根据特定的设备特性定位样式（参见图12.4.3）。要调整网站的呈现样式，让其适应不同的屏幕尺寸，采用媒体查询特别方便。下面列出了可以包含在媒体查询里的媒体特性。

- width（宽度）
- height（高度）
- device-width（设备宽度）
- device-height（设备高度）
- orientation（方向）

- ❑ aspect-ratio（高宽比）
- ❑ device-aspect-ratio（设备高宽比）
- ❑ color（颜色）
- ❑ color-index（颜色数）
- ❑ monochrome（单色）
- ❑ resolution（分辨率）
- ❑ scan（扫描）
- ❑ grid（栅格）

还有一些非标准的媒体特性，如

- ❑ -webkit-device-pixel-ratio（WebKit[①]设备像素比）
- ❑ -moz-device-pixel-ratio（Mozilla[②]设备像素比）

除了 orientation、scan 和 grid 以外，上述属性均可添加 min- 和 max- 前缀。min- 前缀定位的是“大于等于”对应值的目标，而 max- 前缀定位的则是“小于等于”对应值的目标。

在本章里，我们将着重介绍 min-width 和 max-width，因为它们是制作响应式页面时反复用到的两个媒体特性。

现代桌面浏览器和移动电话浏览器对媒体查询的支持程度很高。不过，Internet Explorer 8 及以下版本并不支持媒体查询，相关解决方案请参见 12.6 节。

1. 媒体查询语法和示例

Peter Gasston 的 *The Book of CSS3*（No Starch Press，2011）一书写得很棒。该书对媒体查询的语法做了很好的归纳。以下是媒体查询的基本语法。

- ❑ 指向外部样式表的链接：

```
<link rel="stylesheet" media="logic
→ type and (feature: value)"
→ href="your-stylesheet.css" />
```

- ❑ 位于样式表中的媒体查询：

```
@media logic type and (feature:
→ value) {
 /* 目标CSS样式规则写在这里 */
}
```

我将在随后解释有关的语法，不过几个简单的例子（参见图 12.4.3 和图 12.4.4）有助于在上下文中理解这些语句。示例中的查询是相同的，但它们呈现样式的方式却是不同的。图 12.2.3 中的示例可以翻译为“仅当媒体类型为 screen 且视觉区域最小宽度为 480 像素时，加载并使用 styles-480.css 中的样式规则”。图 12.4.4 中的示例可以翻译为“仅当媒体类型为 screen 且视觉区域最小宽度为 480 像素时，使用下面的样式规则”。（关于**视觉区域**的含义，参见本节最后的补充材料。）对于响应式页面，大多数情况下我们需要将媒体查询放到样式表中。

```
...
<head>
    <meta charset="utf-8" />
    <title>Media query in link</title>
    <meta name="viewport" content="width=
    → device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="base.css"
    → media="all" />

    <!--
    The logic is only.
    The type is screen.
    The feature: value is min-width: 480px.
    -->
    <link rel="stylesheet" media="only
    → screen and (min-width: 480px)"
    → href="styles-480.css" />
</head>
...
```

图 12.4.3　base.css 中的样式用于所有的输出设备。styles-480.css 中的样式则仅用于支持媒体查询且视觉区域宽度不低于 480 像素的浏览器

① WebKit 是 Chrome、Safari 等浏览器使用的网页引擎和 JavaScript 引擎开源程序。——译者注
② Mozilla 是 Firefox 等浏览器的基础程序。——译者注

```
/* 常规样式写在这里。
   每个设备都能获取它们，
   除非被媒体查询中的样式规则覆盖 */
body {
    font: 200%/1.3 sans-serif;
}

p {
    color: green;
}

/*
The logic is only.
The type is screen.
The feature: value is min-width: 480px.
*/
@media only screen and (min-width:
→ 480px) {
    p {
        color: red;
        font-weight: bold;
    }
}
```

图 12.4.4 这个粗略的示例包含了默认的段落样式，接着是当媒体查询结果为真时对段落文本的修改。我将这份样式表保存为 basic-media-query.css，并在如图 12.4.5 所示的页面中加载该文件。结果如图 12.4.6、图 12.4.7 和图 12.4.8 所示

我创建了一个测试页（如图 12.4.5 所示），并指向包含图 12.4.4 中代码的样式表。可以查看该页面在 iPhone（图 12.4.6）、iPad（图 12.4.7）及窄的桌面浏览器（图 12.4.8）中显示的样子。

回到语法，让我们看看这些代码的组成部分。

- *logic*（逻辑）部分是可选的，其值可以是 only 或 not。only 关键字可以确保旧的浏览器不读取余下的媒体查询，同时一并忽略链接的样式表。not 关键字可以对媒体查询的结果求反，让其反面为真。例如，使用 media="not screen" 会在媒体类型为 screen 以外的任何类型时加载样式表。
- *type*（类型）部分是媒体类型，如 screen、print 等。
- *feature*: *value* 对是可选的，但一旦包含它们，它们必须用括号包围且前面要有 and 这个字。*feature* 是预定义的媒体特性，如 min-width、max-width 或者 resolution。对 color、color-index 和 monochrome 特性来说，*value* 是可选的。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Basic media query example</title>
    <meta name="viewport" content="width=
    → device-width, initial-scale=1.0" />
    <link rel="stylesheet"
    href="assets/
    → css/basic-media-query.css" />
</head>
<body>
    <p>Hi, I'm a paragraph. By default, I'm
    → green and normal. But get me in a
    → viewport that's at least 480px wide,
    → and I get red and bold!</p>
</body>
</html>
```

图 12.4.5 这个样式表包含了需要加载的媒体查询，跟其他样式表一样

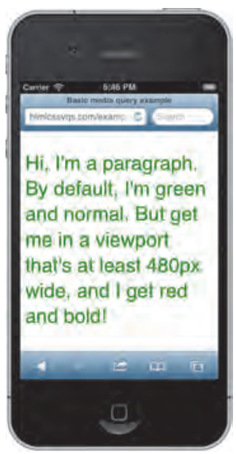


图 12.4.6 在 iPhone 纵向模式下，Mobile Safari 的视觉区域为 320 像素宽，因此文本仍为样式表中基准样式定义的绿色（它从浏览器的默认样式继承了 font-weight 的常规值）。不过，当页面显示在 iPad 中时……

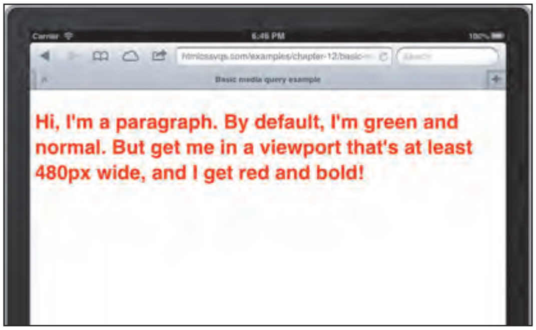


图 12.4.7　……由于在 iPad 纵向模式下浏览器的视觉区域为 768 像素宽，而宽度大于等于 480 像素时会触发媒体查询，文本变成以红色粗体显示。在 iPhone 横向模式下该效果也会生效，因为该模式下视觉区域的宽度刚好为 480 像素

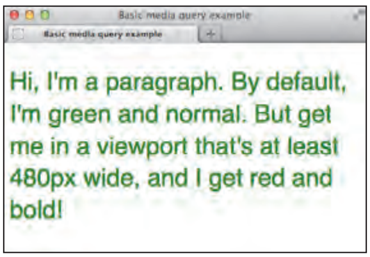


图 12.4.8　现代桌面浏览器也理解媒体查询。这是 Firefox，通过拖拽窗口的右下角使视觉区域的宽度小于 480 像素，因此文本显示为绿色，`font-weight` 为常规值。如果让窗口变大，使其宽度至少为 480 像素，文字将立即以红色粗体显示，无须刷新页面

可以使用 and 将多个特性和值的组合串接起来，还可以创建一系列媒体查询（使用逗号分隔每个媒体查询）。在用逗号分隔的媒体查询列表中，如果有一个媒体查询为真，则整个媒体查询列表为真。图 12.4.9 和图 12.4.10 显示了多种媒体查询。

2. 小结

通过媒体查询，可以根据设备的媒体属性应用不同的样式。尽管媒体查询包含了很多功能，但其中 `min-width` 和 `max-width` 是创建响应式网页时用的最多的。

```
...
    <link rel="stylesheet" media="only
    → screen and (min-width: 480px) and
    → (max-width: 767px)"
    → href="styles.css" />

    <link rel="stylesheet" media="only
    → screen and (orientation:
    → landscape)" href="styles.css" />

    <link rel="stylesheet" media="only print
    → and (color)" href="color-pages.css" />

    <link rel="stylesheet" media="only print
    → and (monochrome)" href="monochrome-
    → pages.css" />

    <link rel="stylesheet" media="only screen
    → and (color), projection and (color)"
    → href="styles.css" />
</head>
<body>
...
```

图 12.4.9　当媒体查询为真时加载外部样式表的示例

在媒体查询中使用 em

到目前为止，本书在媒体查询的示例（参见图 12.4.3、图 12.4.4、图 12.4.9 和图 12.4.10）中均使用像素作为 `min-width` 和 `max-width` 值的单位，这样做有助于帮助读者理解媒体查询的工作原理。不过在实践中，最好使用 em，因为媒体的触发条件往往与访问者浏览器中显示的字体大小有关。Lyza Gardner 对使用 em 的好处作了进一步说明，参见 http://blog.cloudfour.com/the-ems-have-it-proportional-media-queries-ftw/。

在下一部分构建响应式网页的示例页面时，媒体查询将使用 em 值。同时，注释中将说明它们产生的以像素为单位的近似值。

```
/* 基准样式
--------------------------------------- */

/* 针对所有设备的基准样式 */
…


/* 开始媒体查询
--------------------------------------- */
@media only screen and (min-width:
→480px) and (max-width: 767px) {
    /* 样式规则 */
}

@media only screen and (orientation:
→landscape) {
    /* 样式规则 */
}

@media only print and (color) {
    /* 样式规则 */
}

@media only print and (monochrome) {
    /* 样式规则 */
}

@media only screen and (color), projection
→and (color) {
    /* 样式规则 */
}
```

图 12.4.10　这些媒体查询与图 12.4.9 中的是相同的，只是直接出现在样式表中

提示　任何位于媒体查询以外的基准样式规则都会应用于所有的设备。可以使用媒体查询覆盖这些样式规则。需要说明的是，媒体查询样式规则声明仅在与常规样式冲突时进行覆盖，如图 12.4.3 中的 color: green;。如果媒体查询之前的 p 的样式规则包含 font-style: italic;，那么媒体查询为真时段落文本仍以斜体显示，因为媒体查询内的 p 的样式规则并未设置 font-style。

提示　如果你有 Mac，可以使用苹果推出的免费的 iOS Simulator（iOS 模拟器）测试示例页面在 iPhone 和 iPad 中的显示情况。参见 20.1 节。

提示　CSS3 媒体查询规范中列出了关于所有媒体特性的描述（www.w3.org/TR/css3-mediaqueries/#media1）。

理解视觉区域及使用视觉区域 meta 元素

视觉区域（viewport）指的是浏览器（包括桌面浏览器和移动浏览器）显示页面的区域。它不包含浏览器地址栏、按钮这样的东西，只是浏览区域。媒体查询的 width 特性对应的是视觉区域的宽度。不过，device-width 特性不是，它指的是屏幕的宽度。

在移动设备（如 iPhone）上，默认情况下这两个值通常不一样。Mobile Safari（iPhone 的浏览器）的视觉区域默认为 980 像素宽，但 iPhone 的屏幕只有 320 像素宽（iPhone 的 Retina 显示屏的屏幕分辨率有 640 像素宽，但它们是在相同的空间挤入两倍的像素，因此设备宽度仍为 320 像素）。

因此，iPhone 会像设为 980 像素宽的桌面浏览器那样显示页面，并将页面缩小以适应 320 像素的屏幕宽度（在纵向模式下），如图 12.4.11 所示。结果，当你在 Mobile Safari 中浏览大部分为桌面浏览器建立的网站时，会显示将这些网站缩小了的样子。在横向模式下也是这样处理的，只不过宽度为 480 像素（iPhone5 是 568 像素）。如图 12.4.12 所示，如果不进行放大，页面通常是难以阅读的（注意不同设备的默认视觉区域宽度并不相同）。



图 12.4.11　我的测试页面包含一个 320 像素 × 480 像素的绿色 div。Mobile Safari 的视觉区域默认为 980 像素宽，因此 iPhone 会将该 div 缩小，以在 320 像素宽的屏幕内显示它。这就是这个绿色盒子大约占据屏幕宽度的三分之一（即 320/980）的缘故



图 12.4.12　这个测试页面的代码与图 12.4.11 中页面的代码相比，除了包含设置了 width=device-width 的视觉区域 meta 元素以外，其他内容是完全相同的。如你所见，现在的视觉区域宽度与屏幕宽度是相同的

幸好，有一种快速解决方案。在页面的 head 部分添加视觉区域 meta 元素即可。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Your page title</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    ...
</head>
<body>
...
```

这段代码的重点是 width=device-width。有了这条语句，视觉区域的宽度会被设成与设备宽度（对 iPhone 来说为 320 像素）相同的值，因此在纵向模式下该宽度的页面内容会填充整个屏幕（如图 12.4.12 所示）。如果不包含这一语句，使用媒体查询的 min-width 和 max-width 特性将不会得到预期的结果。

代码中的 initial-scale=1. 部分对 width 和 device-width 值没有影响，但通常会包含这一语句。它将页面的默认缩放级别设成了 100%，换成纵向模式也一样。（注意，iOS6 之前的版本有一个 bug，它会裁掉一部分内容，参见 http://adactio.com/journal/5802/。）如果不设置 initial-scale=1，在 iPhone 中，手机从纵向模式改为横向模式时，网页会被放大，从而让布局与纵向模式的一致。

12.5　组合使用

理解了可伸缩图像、弹性布局和媒体查询的知识之后，就可以将它们组合在一起，创建响应式网页。

本书不会演示应用于每个媒体查询块内的全部样式规则。本节将重点关注页面扩张或收缩时切换内容的显示方式所需要考虑的要点。重要的是了解如何建立响应式网站，以及用于实现响应式网站的媒体查询类型。完整的页面和代码见 www.htmlcssvqs.com/8ed/12。

需要说明的是，并不需要先做出一个定宽的设计，再将它转换成响应式的页面。第 11 章演示了如何构建定宽页面，本章（尤其是本节）演示的是如何从零开始构建响应式的页面，就像从未建立过定宽的页面似的。

1. 创建内容和 HTML

一切应该从可靠的、认真考虑过的内容开始。如果你试着用占位符文本（如没有任何含义的lorem ipsum[①]）设计和构建你的网站，当你填入真正的内容以后，你可能会发现形式与内容结合得不好。因此，应该尽可能地将内容采集工作提前，从而对设计和开发满足访问者（和你）需求的网站更有信心。

示例页面的底层 HTML 与第 11 章的页面的代码是相同的，只是我在 `head` 元素中添加了 `<meta name= "viewport" content="width=device-width, initial-scale=1"/>`。关于这行代码的作用，参见上一节的补充材料“理解视觉区域及使用视觉区域 `meta` 元素”。

2. 移动优先方法

我会遵循**移动优先**为页面设计样式，推荐你在设计网页时也遵循这一点。下面是具体过程。

(1) 首先为所有的设备提供基准样式（如图 12.5.1 所示）。这同时也是旧版浏览器和功能比较简单的设备显示的内容。基准样式通常包括基本的文本样式（字体、颜色、大小），内边距、边框、外边距和背景（视情况而定），以及设置可伸缩图像的样式。通常，在这个阶段，需要避免让元素浮动，或对容器设定宽度，因为最小的屏幕并不够宽。内容将按照常规的文档流由上到下进行显示。网站的目标是在单列显示样式中是清晰的、中看的（如图 12.5.2 所示）。这样，网站对所有的设备（无论新旧）都具有可访问性。在不同设备下，外观可能有差异，不过这是在预期之内的，完全可以接受。

(2) 从这种样式开始，使用媒体查询逐渐为更大的屏幕（或其他媒体特性，如 `orientation`）定义样式。大多数时候，`min-width` 和 `max-width` 媒体查询特性是最主要的工具。

这是渐进增强在实战中的应用。（请回顾本书前言中的“渐进增强：一种最佳实践”部分。）处理能力较弱的（通常也是较旧的）设备和浏览器会根据它们能理解的 CSS 显示网站相对简单的版本。处理能力较强的设备和浏览器则显示增强的版本。所有人都能获取到网页的内容。

① 在西方排版、设计领域常用一段以“lorem ipsum”开头的拉丁文作为占位符，以测试排版、设计的效果。这段文本是经过改造的、没有意义的拉丁文，常用“lorem ipsum”指代这段文本。——译者注

```
/* 基准样式
----------------------------------- */
body {
    font: 100%/1.2 Georgia, "Times New
    → Roman", serif;
    margin: 0;
    ...
}

* { /* 参见第11章 */
    -webkit-box-sizing: border-box;
       -moz-box-sizing: border-box;
            box-sizing: border-box;
}

.page {
    margin: 0 auto;
    max-width: 60em; /* 960px */
}

h1 {
    font-family: "Lato", sans-serif;
    font-size: 2.25em; /* 36px/16px */
    font-weight: 300;
    ...
}

.about h2, .mod h2 {
    font-size: .875em; /* 15px/16px */
}

.logo,
.social-sites,
.nav-main li {
    text-align: center;
}

/* 创建可伸缩图像 */
.post-photo, .post-photo-full,
.about img, .map {
    max-width: 100%;
}
...
```

图 12.5.1　应用于所有视觉区域（小屏幕和大屏幕设备）的基准样式示例。这些样式规则与你在本章之前看到的其他代码是类似的，只是它们没有由媒体查询包围。注意我为整个页面设定了 60em 的最大宽度（通常等价于 960 像素），并使用 auto 外边距让其居中。我还让所有的元素使用 box-sizing: border-box;，将大多数图像设置为可伸缩图像

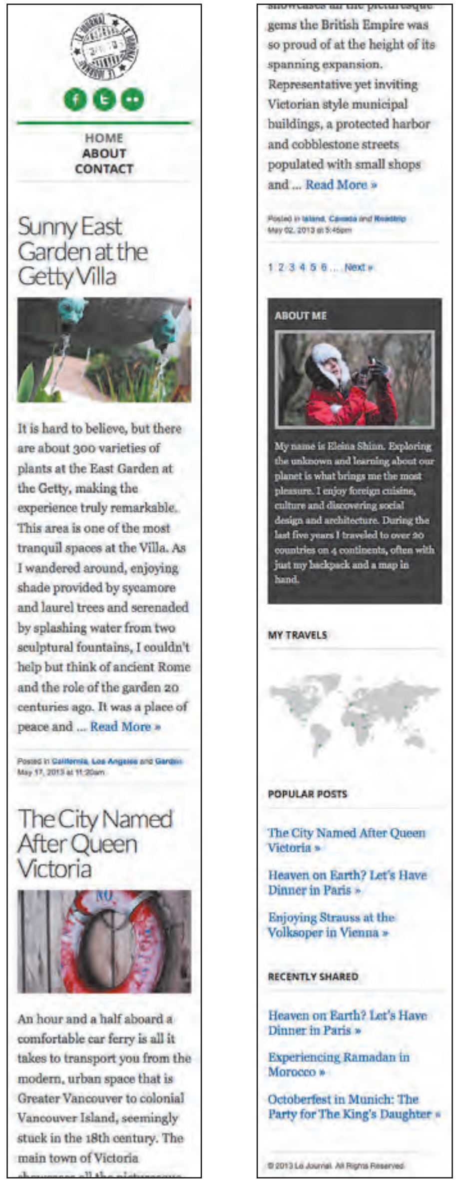


图 12.5.2　仅应用了基础样式的页面的开始点。这个页面在所有的浏览器中都是线性的（图中右侧的部分出现在左侧部分的下面。）此截图展示的是不支持媒体查询的旧浏览器中页面的显示效果。在这种状态下，依然保持了很高的可用性。由于没有设定容器宽度，因此在桌面浏览器中查看页面时，内容的宽度会延伸至整个浏览器窗口的宽度

3. 逐步完善布局

如果用响应式 Web 设计的术语，这一过程可以表述为，使用媒体查询为页面中的每个**断点**（breakpoint）定义样式。断点即内容需作适当调整的宽度。在本例中，应用基准样式规则后，我为下列断点创建了样式规则。记住，对于每个最小宽度（没有对应的最大宽度），样式定位的是所有宽度大于该 `min-width` 值的设备，包括台式机及更早的设备。

- 最小宽度为 20em，通常为 320 像素（如图 12.5.3 和图 12.5.4 所示）。定位纵向模式下的 iPhone、iPod touch、各种 Android 以及其他移动电话。
- 最小宽度为 30em，通常为 480 像素（如图 12.5.5 ~ 图 12.5.8 所示）。定位大一些的移动电话，以及横向模式下的大量 320 像素设备（iPhone、iPod touch 及某些 Android 机型）。
- 最小宽度介于 30em（通常为 480 像素）和 47.9375em（通常为 767 像素）之间，参见图 12.5.9 和图 12.5.10）。这适用于处于横向模式的手机、一些特定尺寸的平板电脑（如 Galaxy Tab 和 Kindle Fire），以及比通常情况更窄的桌面浏览器。
- 最小宽度为 48em，通常为 768 像素（如图 12.5.11 ~ 图 12.5.13 所示）。这适应于常见宽度及更宽的 iPad、其他平板电脑和台式机的浏览器。

```
/* 基准样式
----------------------------------- */
…

/* 20em (大于等于320px)
----------------------------------- */
@media only screen and (min-width:
20em) {
     .nav-main li {
        border-left: 1px solid #c8c8c8;
        display: inline-block;
        text-align: left;
     }

     .nav-main li:first-child {
        border-left: none;
     }

     .nav-main a {
        display: inline-block;
        font-size: 1em;
        padding: .5em .9em .5em 1.15em;
     }
}
```

图 12.5.3 这里针对视觉区域不小于 20em 宽的浏览器修改了主导航的样式。（在假定 body 元素字体大小为 16 像素的情况下，20em 通常等价于 320 像素，因为 20 × 16 ＝ 320。）这样，链接会出现在单独的一行，而不是上下堆叠（如图 12.5.4 所示）。我没有将这些放到基础样式表中，因为有的手机屏幕比较窄，可能会让链接显得很局促，或者分两行显示

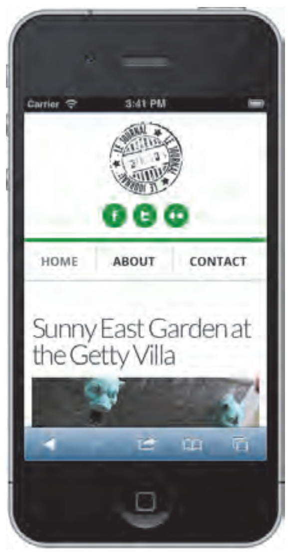


图 12.5.4 根据图 12.5.3 中的媒体查询，主导航显示为一行，每个链接之间由灰色的竖线分隔。这个样式会在 iPhone（以及很多其他的手机）中生效，因为它们在纵向模式下是 320 像素宽。如果你希望报头更矮一些，可以让标识居左，社交图标居右。不过我将这种样式用在下一个媒体查询中（参见图 12.5.5 和图 12.5.6）

```
/* 基准样式
---------------------------------- */
...

/* 20em （大于等于320px）
---------------------------------- */
@media only screen and (min-width: 20em) {
     ...
}

/* 30em （大于等于480px）
---------------------------------- */
@media only screen and (min-width: 30em) {
     .masthead { position: relative; }

     .social-sites {
        position: absolute;
        right: -3px;
        top: 41px;
     }

     .logo {
        margin-bottom: 8px;
        text-align: left;
     }

     .nav-main {
        margin-top: 0;
     }
}
```

图 12.5.5　现在，样式表中有了定位视觉区域至少为 30em（通常为 480 像素）的设备的媒体查询。这样的设备包括屏幕更大的手机，以及横向模式下的 iPhone（参见图 12.5.6）。这些样式会再次调整报头。前两个规则跟第 11 章使用的规则一样，效果也一样

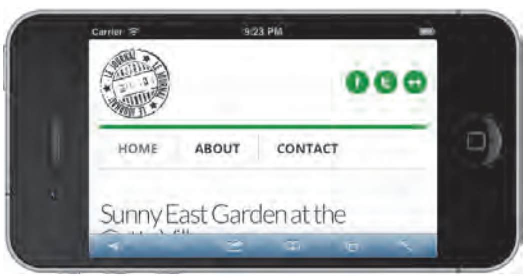


图 12.5.6　这是在 30em 宽的屏幕下显示页面顶部的样子，报头是完整的。在更大的视觉区域，报头宽度会自动调大

```
...

/* 30em（大于等于480px）
---------------------------------- */
@media only screen and (min-width: 30em) {

     ... 报头样式 ...

     .post-photo {
        float: left;
        margin-bottom: 2px;
        margin-right: 22px;
        max-width: 61.667%;
     }

     .post-footer {
        clear: left;
     }

}
```

图 12.5.7　继续在同一个媒体查询块内添加样式，让图像向左浮动，并减少其 max-width，从而让更多的文字可以浮动到其右侧。11.12 节解释了为什么要在博客摘要下面的页脚上设置 clear: left;



图 12.5.8　文本环绕在浮动图像周围

你的断点可能跟此处用的不同。这取决于哪些断点适合你的内容和设计。为适应更宽的视觉区域，一般不会创建超过 48em 的断点。你也不一定要严格按照设备视觉区域的宽度创建断点。如果一个基于 (min-width: 36em) 的断点非常适合你的内容，就可以大胆地使用这个断点。

```
/* 基准样式
----------------------------------- */
...

/* 20em（大于等于320px）
----------------------------------- */
@media only screen and (min-width: 20em) {
     ...
}

/* 30em（大于等于480px）
----------------------------------- */
@media only screen and (min-width: 30em) {
     ...
}

/* 30em - 47.9375em
     (在480px和767px之间)
----------------------------------- */
@media only screen and (min-width: 30em)
→ and (max-width: 47.9375em) {
     .about { /* self-clear float */
        overflow: hidden;
     }

     .about img {
        float: left;
        margin-right: 15px;
     }
}
```

图 12.5.9　现在，让 About Me 图像向左浮动。不过，这种样式仅当视觉区域的宽度在 30em（如图 12.5.10 所示）和 47.9375em 之间时才生效。超过这个宽度会让布局变成两列布局，About Me 文字会再次出现在图像的下面。这个随后实现



图 12.5.10　浮动的 About Me 图像已显示为其本来的尺寸（270 像素宽），它旁边的空间太小，无法很好地容纳文本。这就是之前减少其 `max-width` 的原因（如图 12.5.7 和图 12.5.8 所示）

```
/* 基准样式
----------------------------------- */
...

/* 20em（大于等于320px）
----------------------------------- */
@media only screen and (min-width: 20em) {
     ...
}

/* 30em（大于等于480px）
----------------------------------- */
@media only screen and (min-width: 30em) {
     ...
}

/* 30em～47.9375em
     （在480px和767px之间）
----------------------------------- */
@media only screen and (min-width: 30em) and
→ (max-width: 47.9375em) {
     ...
}

/* 48em（大于等于768px）
----------------------------------- */
@media only screen and (min-width: 48em) {
     .container {
        background: url(../img/bg.png)
        → repeat-y 65.9375% 0;
        padding-bottom: 1.875em;
     }

     main {
        float: left;
        width: 62.5%; /* 600px/960px */
     }

     .sidebar {
        float: right;
        margin-top: 1.875em;
        width: 31.25%; /* 300px/960px */
     }

     .nav-main { margin-bottom: 0; }
}
```

图 12.5.11　这是最终的媒体查询，定位至少有 48em 宽的视觉区域。该媒体查询对大多数桌面浏览器（参见图 12.5.13）来说都为真，除非用户让窗口变窄。它同时也适用于纵向模式下的 iPad 及其他一些平板电脑，如图 12.5.12 所示



图 12.5.12　页面变得完整了。这里显示的是在 iPad 中页面显示的样子，在桌面浏览器中（尽管要宽一些）也是类似的。由于宽度是用百分数定义的，因此主体内容栏和附注栏会自动伸展



图 12.5.13　在一个至少有 960 像素宽的浏览器中的样子。看起来很像我们在第 11 章中创建的定宽布局。如果让浏览器的宽度变窄，布局就会自动变换为移动设备上应该显示的样子

4. 构建响应式 Web 页面

(1) 创建内容和 HTML。

(2) 在 HTML 页面的 `head` 元素中，输入 `<meta name="viewport" content="width=device-width" />` 或 `<meta name="viewport" content="width= device-width, initial-scale=1" />`。

(3) 创建适用于所有设备的基准样式（参见“移动优先方法”）。确保页面中的图像可伸缩，参见 12.2 节。

(4) 识别出适合你的内容的断点。创建相关的媒体查询，让布局适应从小屏幕到大屏幕的不同可视区域宽度。（参见“逐步完善布局”）。

(5) 如果需要为第 (4) 步中页面的一些内容指定宽度，使用百分数（参见 12.3 节）。

(6) 选择你希望的旧版 IE 显示页面的方式（参见 12.6 节）。

(7) 开始测试（参见“测试响应式网页”）。

(8) 根据需要，修改第 (3) ~ (5) 步中的 CSS，并进行测试，直到页面在各种设备下都呈现出预期的效果。

提 示　Eivind Uggedal 的 http://mediaqueri.es 网站汇集了大量现实的响应式网站（数量还在增加），值得借鉴。

提 示　Screen Sizes 网站（http://screensiz.es）提供了流行设备和显示屏的分辨率和设备宽度信息。使用媒体查询的时候，这些信息可能很有用。

提 示　Maximiliano Firtman 维护了一个现代移动设备对 HTML5 和 CSS3 支持情况的表格，见 http://mobilehtml5.org。（其中大量信息属于 HTML5 高级特性，本书并未涉及。）

测试响应式页面

你可能希望在发布响应式页面之前先在移动设备和桌面浏览器上对其测试一遍。

构建响应式页面的时候，你可以放大或缩小桌面浏览器的窗口，模拟不同手机和平板电脑的视觉区域尺寸。然后再对样式进行相应的调整。这当然是一种不够精细的办法，但它确实有助于建立有效的样式，从而减少在真实设备上优化网站的时间。Malte Wasserman 提供的 Viewport Resizer（http://lab.maltewassermann.com/viewport-resizer/）对这一工作很有帮助。

不过不要就此打住。对你的浏览器多进行几次放大或缩小，测试布局在桌面浏览器各种宽度下如何调整显示。你可能会找到其他需要引入额外媒体查询的断点。

20.2 节总结了对移动和桌面测试的方法。

对 Retina 及类似显示屏使用媒体查询

有时你可能希望为高像素密度设备设定样式（参见 5.8 节）。一种常见的用例是为这样的显示屏准备双倍尺寸（2x）的 sprite，从而让图像显得更锋利。（参见 14.10 节。）

假设你的 sprite 的原始尺寸是 200 像素 ×150 像素，其中每个图像都用 1 像素分隔。创建一个双倍大小的版本（400 像素 ×300 像素），每个图像之间就有 2 像素的间隔。同时，sprite 中的每个图像都是原始尺寸的 2 倍。针对高像素密度设备，可以使用下面的媒体查询：

```
@media (-o-min-device-pixel-ratio: 5/4),
(-webkit-min-device-pixel-ratio:1.25),
(min-resolution: 120dpi) {
    .your-class {
        background-image:url(sprite-2x.png);
        background-size: 200px 150px;
    }
}
```

注意 background-size 设置成了原始尺寸，而不是 400 像素 ×300 像素。这样会让图像缩小，为原始尺寸创建的样式对 2x 版本也有效。

12.6 兼容旧版 IE

对于移动优先方法，有一点需要注意，就是 Internet Explorer 8 及以下的版本不支持媒体查询。这意味着这些浏览器只会呈现媒体查询以外的样式，即基准样式。目前，在世界上大部分地区，使用 IE6 和 IE7 的用户已经非常少了。因此，真正费脑筋去考虑的是 IE8，它在全世界所占的份额不到 9%，且这个数字还在下降（详情参见 http://gs.statcounter.com）。

对于 IE8（及更早的版本），有三种解决方法。

- 什么都不做。让网站显示基本的版本。
- 为它们单独创建一个样式表，让它们显示网站最宽的版本（不会形成响应

式的网页）。一种做法是复制一份常规的样式表，将其命名为 old-ie.css 之类的文件名。将媒体查询语句去掉，但保留其中的样式规则。在 HTML 中添加条件注释，从而让不同的浏览器都能找到正确的样式表（参见图 12.6.1）。

- 如果希望页面有响应式的效果，就在页面中引入 respond.min.js（参见图 12.6.2）。Scott Jehl 创建了这段简短的代码，它让 `min-width` 和 `max-width` 媒体查询对旧版 IE 也有效。参见补充材料。

```
...
<head>
    ...

    <!--[if gt IE 8]><!-->
    <link rel="stylesheet"
    → href="css/styles.css" />
    <!--<![endif]-->

    <!--[if lt IE 9]>
    <link rel="stylesheet"
    → href="css/old-ie.css" />
    <![endif]-->
</head>
...
```

图 12.6.1　这里有很多看起来很怪异的代码。你可能不熟悉的代码是条件注释（参见 www.quirksmode.org/css/condcom.html）。第一个条件注释包含的是为 IE8 及更旧版本以外的所有浏览器准备的样式表。第二个包含的则是仅为 IE9 以下的 IE 准备的样式表

```
...
<head>
...
    <link rel="stylesheet"
    → href="css/styles.css" />

    <!--[if lt IE 9]>
    <script src="js/respond.min.js">
    → </script>
    <![endif]-->
</head>
...
```

图 12.6.2　将 `src` 值里的 `js` 部分替换为你网站中的地址（如果跟这里的不一样的话）。设置好以后，IE8 及以下版本会理解 `min-width` 和 `max-width` 媒体查询，并呈现相应的样式。这样做的话，就没有必要将 IE 样式表分离出来了。这个 `script` 元素外围的条件注释是可选的，不过如果包含的话，就只有 IE8 及以下版本会加载 respond.min.js

第二种方法更可靠，它让 IE8 用户也能看到网站的完整布局。

提 示　如果使用 Sass、LESS 或 Stylus 这类 CSS 预处理器，就可以自动创建为旧版 IE 准备的样式表。Nicolas Gallagher 分享了一个使用 Sass 的方法（http://nicolasgallagher.com/mobile-first-css-sass-and-ie/）。

使用 Respond.js

可以访问 https://github.com/scottjehl/Respond，点击 ZIP 按钮下载 Respond.js。下载到电脑后，打开该 zip 文件，然后将 respond.min.js 复制到你网站的一个文件夹里。如果这段脚本不起作用，不妨看看上述 GitHub 链接中的 Support & Caveats（支持与注意事项）部分。Respond.js 并不是一个适用于所有网站的完美方案。

第 13 章

使用 Web 字体

13

本章内容

- 什么是 Web 字体
- 在哪里能找到 Web 字体
- 下载第一个 Web 字体
- 理解 `@font-face` 规则
- 为 Web 字体添加样式
- 为 Web 字体应用斜体和粗体
- 使用 Google Fonts 的 Web 字体

过去几年，我们看到了在万维网上使用字体的复兴。过去，我们通常只能在非常有限的字体集中进行选择。我们在第 10 章中提到过，最好使用那些用户很可能在自己的电脑上安装了的字体。这也是大多数网站将 `body` 字体设为 Arial、Georgia、Verdana 或 Trebuchet MS 的原因。

现在有了 Web 字体，我们在开发 Web 项目时有了大量字体可供选择。因此，Web 设计师可以为访问者提供更为精细的体验。经历这种改变既让人着迷，又令人激动。

本章解释了 Web 字体的基础，如何使用自托管的 Web 字体（例如 Font Squirrel 提供的字体），如何使用 Google Font。学会如何使用自托管的字体需要花费较多的时间。如果你想在自己的网站上尽快用上 Web 字体，可以阅读 13.7 节。

13.1　什么是 Web 字体

CSS 规则 `@font-face` 为 Web 字体创造了可能，该样式规则允许 CSS 链接到服务器上的一种字体供网页使用。

很多人认为 Web 字体是新生事物。实际上，Web 字体大约在 1998 年就产生了。Netscape Navigator 4 和 Internet Explorer 4 均采用了这种技术，但它们的实现都不支持标准的字体文件格式，因此很少有人用到它们。直到将近十年以后，浏览器才开始采纳这种使用更为常见的字体文件格式的标准，Web 字体的使用才变得常见起来。

1. Web 字体文件格式和浏览器支持

Web 字体可以使用一系列文件类型。需要特别指出的是，下面介绍的前三种字体类型是如今经常使用的。

- 内嵌 OpenType（Embedded OpenType，.eot）。在使用 `@font-face` 时，Internet Explorer 8 及之前的版本仅支持内嵌 OpenType。内嵌 OpenType 是 Microsoft 的一项专有格式，它使用数字版权管理技术防止在未经许可的情况下使用字体。

- TrueType（.ttf）和 OpenType（.otf），台式机使用的标准字体文件类型。TrueType 和 OpenType 得到了 Mozilla Firefox（3.5 及之后的版本）、Opera（10 及之后的版本）、Safari（3.1 及之后的版本）、Mobile Safari（iOS 4.2 及之后的版本）、Google Chrome（4.0 及之后的版本）及 Internet Explorer（9 及之后的版本）的广泛支持。这些格式不使用数字版权管理。
- Web 开放字体格式（Web Open Font Format，.woff）。这种较新的标准是专为 Web 字体设计的。Web 开放字体格式的字体是经压缩的 TrueType 字体或 OpenType 字体。WOFF 格式还允许在文件上附加额外的元数据。字体设计人员或厂商可以利用这些元数据，在原字体信息的基础上，添加额外的许可证或其他信息。这些元数据不会以任何方式影响字体的表现，但经用户请求，这些元数据可以呈现出来。Mozilla Firefox（3.6 及之后的版本）、Opera（11.1 及之后的版本）、Safari（5.1 及之后的版本）、Google Chrome（6.0 及之后的版本）及 Internet Explorer（9 及之后的版本）均支持 Web 开放字体格式。
- 可缩放矢量图形（Scalable Vector Graphics，.svg）。简言之，应避免对 Web 字体文件使用 SVG。它更多地用于早期 Web 字体，因为它是 iOS 4.1 上移动 Safari 唯一支持的格式（还有可能引起一些崩溃的情况）。从 iOS 4.2（于 2011 年初即被广泛使用）起，移动 Safari 开始支持 TrueType。

可以看到，Web 字体的跨平台支持程度是很高的，因此没有任何理由不在今天的网站上使用它们。

2. 法律问题

从技术层面上说，字体都是小的软件。我认识以字体设计和制作为生的人，这是一种艰苦、细致的创造性活动，没有强大的心智是做不了这份工作的。

出于这一原因，即便 `@font-face` 功能从一开始就存在，它还是会激怒一些人也就不难理解了。毕竟，如果浏览器能链接并下载某种字体，意味着任何人都能下载并在自己的计算机上安装这种字体，无论他们是否购买了这种字体。

这就是为什么 Web 设计人员和开发人员必须确保网站上用到的任何字体都具有在万维网上使用的许可。大多数厂商和字体服务提供商通过将许可作为字体购买的一部分或菜单选项提供这种许可。

不管选用哪种方式，确保对项目中使用的 Web 字体具有稳定的权利。为此，你可以查看购买的字体的许可证。有些字体可以在 Adobe Photoshop、Adobe Illustrator 等桌面应用软件中使用，但不能在网站上作为 Web 字体使用。你要购买字体的厂商的网站通常会提到这些信息。如果存在疑惑，可以跟厂商联系，了解哪些权利是允许的。

如果你购买了一个字体，并确切地知道可以将其用做 Web 字体，就可以使用 Font Squirrel 提供的免费 `@font-face` 生成器（www.fontsquirrel.com/fontface/generator）。该工具可以将字体转化为在万维网上使用的所有 Web 字体文件类型。

真正的好消息是 Web 字体服务的大量涌现。通过它们，可以在网站上使用多达数千种的 Web 字体（参见 13.2 节），其中很多还是免费的。

3. 管理文件尺寸

使用 Web 字体，尤其是使用多种 Web 字体的一个潜在风险是它们可能会增加页面的“重量”。我在这里谈论的不是炸薯条和甜甜圈，我谈论的是千字节和兆字节。

对于所有这些字体来说，在它们显示到页面上之前，都需要下载到用户的计算机上。这会减慢网站加载的速度，尤其是对移动用户来说。我的建议是审慎地选择 Web 字体。如果你发现自己使用了好几个 Web 字体，就应该想办法对这些字体进行整合了。

◎ 构造子集

节省页面重量的一种方法是**构造子集**（subsetting）。构造子集是通过仅包含确定使用的字符削减实际字体大小的方法。例如，如果对标题使用 League Gothic 字体，而网站的设计要求标题始终使用大写字母，没有使用小写字母的需要。通过构造子集，可以将小写字母从字体中移除，这样字体文件大小就会减小一些。

此外，对于很多字体，你还可以选择针对特定语言的子集。13.3 节解释了如何在 Font Squirrel 中采用这种方法。

解释子集构造的具体细节已经超出了本书的范围，不过 Font Squirrel 提供的 `@font-face` 生成器（之前提到过）可以帮助你进行专家级的子集构造。

13.2 在哪里能找到 Web 字体

在网站上使用 Web 字体有两种方式：自托管和 Web 字体服务。这两种方式都是完全有效的选项，不过二者差异很大，各有利弊。当你权衡这些利弊的时候，你会发现并非所有的 Web 字体都是随处可见的。你可能会发现，即便你想采用自托管的方式，你需要的字体也只能来自 Web 字体服务。你还有可能会发现你希望使用的服务并未提供你想要的某种字体。这时，你可能需要寻找一种接近的替代方案，或者重新考虑你的方法。在作出决定之前，需要花费一些精力权衡所有的选项并灵活处理。

1. 自托管

自托管 Web 字体来源于你自己的服务器，就像其他的资源（如图像、CSS 文件）一样。如果需要与字体相关的花费，通常也是一次性的购买支出，是否将字体文件上传到网站上并将其包含到代码中取决于你自己。在接下来的小节，我们会讲解使用自托管服务（使用的是从 Font Squirrel 下载的一款免费 Web 字体）的详细步骤。

寻找可以自托管的字体是比较容易的，因为它们的数量很多。它们的质量和价格差异很大（有的甚至是免费的）。其中一些流行的自托管字体来源如下。

- ❑ Font Squirrel（www.fontsquirrel.com），参见图 13.2.1



图 13.2.1 Font Squirrel 是寻找可以自行托管的免费 Web 字体的一个流行的网站。在本章接下来的小节里，你将学会如何使用它。你北非必须要了解使用 Google Fonts 这样的 Web 字体服务的全部细节，但了解它们有助于理解 Web 字体的工作原理，并了解一些常见的陷阱

- ❑ MyFonts（http://myfonts.com）
- ❑ The League of Moveable Type（www.theleagueofmoveabletype.com），免费
- ❑ FontShop（www.fontshop.com）

2. Web 字体服务

Web 字体服务通常提供订购 Web 字体的方法。这种方式是指按月或按年支付使用字体的版权费用，而不是彻底买断字体。Typekit 是这个领域的先驱，现在已经存在几种服务了。

这些服务托管字体，为用户提供一小段代码放在其网页中。这段代码可能是 JavaScript，也可能是 CSS，这取决于服务自身。它包含了从远程服务器获取字体文件并将其显示在网站上这一过程所需的全部代码。很多人主张使用这种方式，因为这种方式通常比单独购买字体更便宜，而且它让用户有机会尝试很多种不同的字体。

一些流行的 Web 字体服务包括：

- ❑ Cloud.typography（www.typography.com/cloud/welcome/）
- ❑ Edge Web Fonts（www.edgefonts.com），免费
- ❑ Fontdeck（http://fontdeck.com）
- ❑ Fonts.com（www.fonts.com/web-fonts）
- ❑ Fontspring.com（www.fontspring.com）
- ❑ Google Fonts（www.google.com/fonts），免费，参见图 13.2.2
- ❑ Typekit（https://typekit.com）
- ❑ WebINK（www.webink.com）

按理说，Web 字体服务可以提供比自托管更多的特性。如果出现了更好的字体文件或代码，服务便可以很容易地将它们提供给用户。例如，Typekit 中的一些字体使用基于 PostScript 的轮廓，在 Windows 浏览器上显示这些字体时，Typekit 会提高它们的品质，让它们的边缘看起来更平滑。

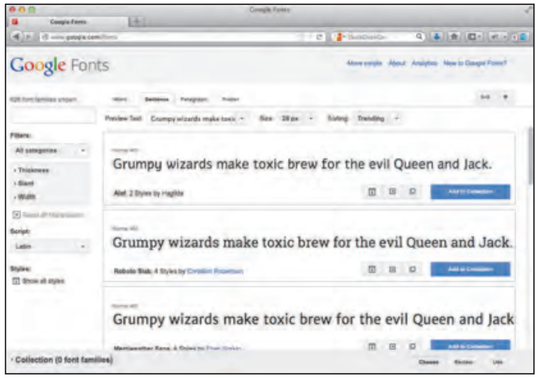


图 13.2.2　Google Fonts 是一种流行的免费 Web 字体服务。我们将在 13.7 节学习如何使用它

在这类服务中，很多都使用 JavaScript 嵌入获取 Web 字体的代码。这样做有一些好处，也有一些不好的地方。在这一过程中，JavaScript 做了不少工作，包括微调 Web 字体显示设置，为字体本身的加载提供额外的控制。这种控制可以产生更好的体验。

这种奢华的代价当然是你必须百分之百地依赖 JavaScript，而 JavaScript 会影响页面的性能。在 Web 字体呈现在页面上之前，用户必须等待 JavaScript 加载完成。这是你决定如何在网站上加入 Web 字体时需要记住的事情。不过别因此拒绝使用 Web 字体服务，很多人，包括我自己，使用 Web 字体都很顺手。

3. Web 字体的质量与显示

不幸的是，并非所有的 Web 字体都是一样的。不同浏览器中 Web 字体显示的效果具有很明显的差异。一些字体在 Internet Explorer 较早的版本中显示效果并不太好，这一点尤为明显。

此外，有些字体在某些字体大小下的显示效果要好于其他字体大小下的效果。它们有可能过于纤细，不适合用来做正文字体，或者用做标题时缺乏权重。

当你选择字体时，要尽量检查你即将选择的字体在各种浏览器中显示的效果。现在，由于很多 Web 字体公司提供 Web 字体的实时示例，有的公司提供字体在不同浏览器和平台中的截图，因此上述工作就变得容易多了。

如果你坚持自己做这些测试，可以试试 Web Font Specimen（http://webfontspecimen.com）提供的资源。这是一个测试 Web 字体在各种环境和字体大小下的显示效果的工具。

4. 那么，如何开始呢

本章接下来的章节将关注两个方面的内容：如何使用自托管的 Web 字体（参见图 13.2.1）和如何使用 Google Fonts（参见图 13.2.2）。其中，Google Fonts 更为简单直接，因此你可以先从那里开始。

图标字体以及如何获取

图标字体指的是包含图标而非字母、数字和标点符号的字体。你可以像对待文字那样对它们添加样式，如设置颜色。

图标字体最棒的一点是，无论使用什么样的字体大小，它们都会平滑缩放，边缘总是很锐利。因此对图标图像来说这是一个很好的选择，因为无需再为让它们在不同类型的显示屏（包括 Retina 显示屏）上或应用不同字体大小的响应式页面上都清晰显示而做额外的工作。

在第 11 章和第 12 章的示例页面的报头中，我使用 Fontfabric（www.fontfabric.com）提供的 Socialico 创建了社交网站图标。通过将其上传到 Font Squirrel 的 Webfont Generator 创建了页面所需的 Web 字体格式。

Chris Coyier 收集了大量获取图标字体的资源（参见 http://css-tricks.com/flat-icons-icon-fonts/），还演示了如何使用 IcoMoon（http://icomoon.io）创建开发人员自己的图标字体（参见 http://css-tricks.com/video-screencasts/113-creating-and-using-a-custom-icon-font/）。

13.3 下载第一个 Web 字体

在页面中使用自托管的 Web 字体为元素添加样式时，需要先将 Web 字体文件下载下来。下载免费的 Web 字体很快也很方便。我们将使用 Font Squirrel，不过使用其他字体服务提供商所需的操作与下面的步骤都是相似的。

1. 从 Font Squirrel 下载 Web 字体

(1) 访问 Font Squirrel（www.fontsquirrel.com），选择一种你希望使用的字体。有很多浏览字体的方式，如通过主页，通过 Popular（流行）或 Recent（最新）板块，或者通过搜索。这里选择了 PT Sans（如图 13.3.1 所示）。



图 13.3.1 通过点击字体的名称（如图所示）选择字体，而不要使用右边的蓝色下载按钮（图中未显示）。那个下载链接用于在你的系统中使用对应的字体，而非用于下载 Web 字体文件

(2) 在字体所在的页面选择 Webfont Kit（如图 13.3.2 所示）。

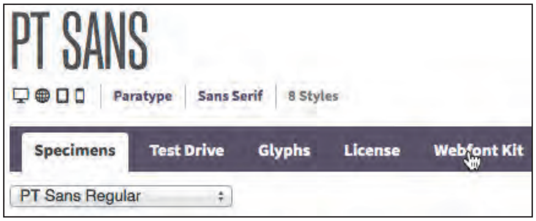


图 13.3.2　Specimens 部分让你可以看到字体在常规、加粗、斜体等文本样式（取决于字体提供的样式）下的字体效果。可以在 Test Drive 部分输入一些文本并查看效果。Glyphs 部分会显示字体中的每个字符，而 License 部分则会说明哪些使用情形是允许的。如果你看上了某款字体，请进入 Webfont Kit

(3) 在 Choose Font Formats（选择字体格式）中，取消 SVG 选项（如图 13.3.3 所示）。如果你需要让使用 Internet Explorer 8（或更早版本）的访问者看到你的 Web 字体，则需要选择 EOT，否则不选择该项。

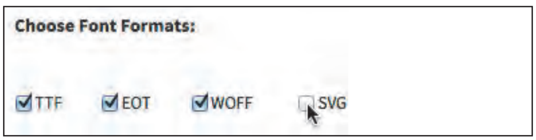


图 13.3.3　SVG 不再是推荐的 Web 字体格式，因此这里对它取消选择。如果你认为访问你的网站的 IE8 用户不多，则不必选择 EOT

(4) 选择字体的子集。通常，要么使用默认选项，要么选择网站内容所使用的语言（这里选择 English，如图 13.3.4 所示）。

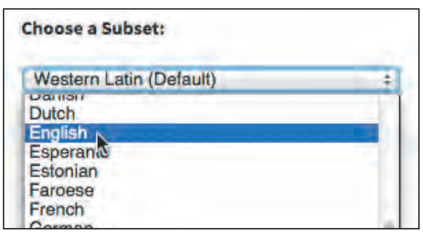


图 13.3.4　对一种 Web 字体构造子集可以减少其包含的字符数量，从而减小文件大小

(5) 点击 DOWNLOAD @FONT-FACE KIT 按钮（如图 13.3.5 所示），下载应该会立即开始。下载的文件为 ZIP 包。



图 13.3.5　将字体下载到你的电脑

(6) 下载完成后，打开压缩包，应该看到一个名为 web fonts 的文件里，里面至少有一个字体文件夹（如图 13.3.6 所示）。

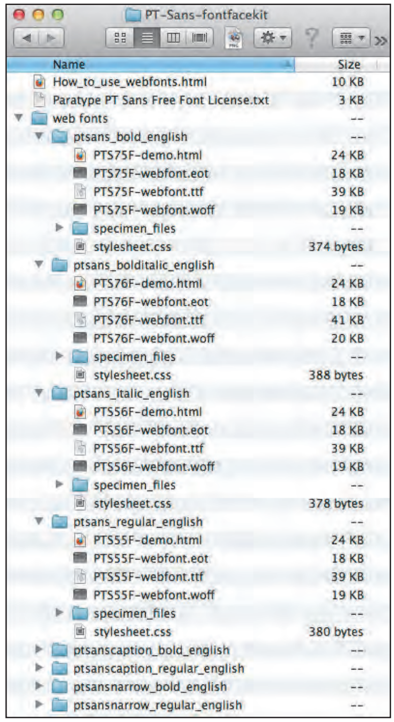


图 13.3.6　天哪，怎么会有这么多文件和文件夹！为什么有这么多呢？在第 (3) 步（如图 13.3.3 所示）中选择的字体不同，这里看到的文件和文件夹数量也不一样。如果在 Font Squirrel 的 PT Sans 页面中选择 Specimens 标签页（如图 13.3.2 所示），将会看到 8 种字体（如图 13.3.7 所示）。它们代表了 PT Sans 的不同样式和粗细版本。我们选择了三种格式，因此 ZIP 文件里就有八种字体的文件夹，每个文件夹里都有各自的三种格式的字体文件，以及示例 HTML 和 stylesheet.css 文件。在本章中，我仅使用图中显示的四个展开的文件夹中的字体文件以及 stylesheet.css 中的部分代码



图 13.3.7　PT Sans 有三个版本，共 8 种样式：PT Sans（常规、加粗、斜体和加粗斜体）、PT Sans Caption（常规、加粗）以及 Pt Sans Narrow（常规、加粗）。本章不会用到 PT Sans Caption 或 PT Sans Narrow。有些字体只有很少的几种样式，有的甚至只有一种样式（如 Bebas Neue）

2. 在 demo.html 文件中查看所选字体

Web 字体文件夹中的每个文件夹都有一个演示该字体的 HTML 页面。其文件名均以 -demo.html 结尾。在浏览器中打开该文件，如图 13.3.8 所示（参见 2.7 节）。



图 13.3.8　从 Font Squirrel 下载的 PT Sans 字体包中的 PTS75F-demo.html 文件。这个文件位于 ptsans_bold_english 文件夹。如果要查看其他的 PT Sans 样式，就打开对应的文件夹中的演示文件（例如 ptsans_regular_english 文件夹中的 PTS55F-demo.html）。每个文件夹里都包含一个 stylesheet.css

演示文件显示 Web 字体是有效的。这很让人激动！在你欢庆胜利之前，我们将在下一节进一步探讨其工作原理，以及如何将它用到你自己的页面中去。

提示　通常，一个页面中不应使用两个（最多三个）以上的 Web 字体，因为引用的字体文件越多，访问者浏览器要下载的文件就越多。这样会减慢页面加载和呈现的速度，特别是对于使用智能手机等设备（意味着较慢的网络连接）的访问者，体验就更糟糕了。需要说明的是，这里说的数量指的是一种或多种字体的两种或三种样式或粗细版本。例如，PT Sans 常规、加粗和斜体算三种，因此如果你引入了另一种 Web 字体系列，则至少引入了四种字体。

提示　你开始下一个项目之前，想获得选择字体的灵感吗？ Typekit 团队写了一篇很好的博客文章，这篇文章介绍了大量关于 Web 字体及一般性排版的有用信息。初学者可以尝试阅读 Sites we like（我们喜爱的网站）系列，参见 http://blog.typekit.com/category/sites-we-like/。

提示　需要在 Photoshop 中使用这些字体吗？你可以将 Web 字体包中的 TrueType（.ttf）字体装到你自己的计算机上。安装完成后就可以像使用其他字体一样使用该字体了。

13.4　理解 @font-face 规则

现在，是时候看看 Web 字体的工作原理了。让我们看看前四个 PT Sans 字体文件夹中的 stylesheet.css 文件。每个文件都有一条为一种 PT Sans 样式准备的规则，位于看起来有些奇怪的 `@font-face { ... }` 中（如图 13.4.1 所示）。

```
@font-face {
    font-family: 'pt_sansregular';
    src: url('PTS55F-webfont.eot');
    src: url('PTS55F-webfont.eot?#iefix')
        ➝ format('embedded-opentype'),
        url('PTS55F-webfont.woff')
        ➝ format('woff'),
        url('PTS55F-webfont.ttf')
        ➝ format('truetype');
    font-weight: normal;
    font-style: normal;
}

@font-face {
    font-family: 'pt_sansitalic';
    src: url('PTS56F-webfont.eot');
    src: url('PTS56F-webfont.eot?#iefix')
        ➝ format('embedded-opentype'),
        url('PTS56F-webfont.woff')
        ➝ format('woff'),
        url('PTS56F-webfont.ttf')
        ➝ format('truetype');
    font-weight: normal;
    font-style: normal;
}

@font-face {
    font-family: 'pt_sansbold';
    src: url('PTS75F-webfont.eot');
    ...
}

@font-face {
    font-family: 'pt_sansbold_italic';
    src: url('PTS76F-webfont.eot');
    ...
}

...
```

图 13.4.1　这些样式来自四个 PT Sans 的 stylesheet.css 文件中的第一个。（我改变了样式规则的顺序，从而与本章介绍的顺序一致。）这些代码块都是一样的，只有突出显示的部分不一样。不一样的地方就是 font-family 值，即每种 Web 字体文件名的前缀（注意：你可能注意到在你的 stylesheet.css 中有一个额外的引用 SVG 的 URL。我在这里忽略了这一行代码，因为我没有下载该格式）

这段代码的语法与传统的 CSS 有些不同，因为 @font-face 部分并不是你要添加样式的元素的选择器。因此，这段样式不会影响任何元素的样式，而是让样式表知道 Web 字体的存在，从而可以在其他的样式规则中用它对文本设置样式。（每个文件夹的演示 HTML 文件都会加载 stylesheet.css 并包含使用该字体的 CSS 规则。）

每条规则的第二行是为字体系列准备的，例如，font-family: 'pt_sansregular'; 或 font-family: 'pt_sansbold';。

这建立了使用某种特定的 Web 字体的名称——对元素设置字体时需要用到的名称，就像常规的字体一样。这个名称可以使用你选择的任何名称。除了可以使用 pt_sansregular 以外，你还可以选择 Banana 或 The Best Font Ever，具体情况取决于个人。事实上，我们在下一节为页面应用 PT Sans 字体时就会改变这个名称。

这条样式规则的下面几行告诉浏览器字体文件的位置。其中包括为所有支持 Web 字体的不同浏览器准备的字体文件格式。语法看起来有些吓人，但我们的目的并不是要完全理解它们。如果你想进一步深入研究，了解这样显示的原理，我向你推荐 Ethan Dunham 发表在 Fontspring 上的博文，参见 www.fontspring.com/blog/further-hardening-of-the-bulletproof-syntax。

13.5　使用 Web 字体设置文本样式

我们讲过了 @font-face 的语法，但还从来没有真正地将 Web 字体放到网页里去。尽管我们将要使用的是从 Font Squirrel 下载的字体，但这些方法也同样适用于其他任何自托管的字体。

在样式表中使用 Web 字体有多种方法。其中的一种便是 Font Squirrel 使用的方法，我

们可以通过查看 stylesheet.css 和演示 HTML 文件了解这种方法。这里将讲解另外一种方法。

没有哪种方法完全正确或完全错误，不过个人推荐我们即将介绍的方法。它更接近你所习惯的使用常规字体为文本设置样式的方式，也更方便管理（尤其是对于刚接触 Web 字体的人来说），而且还能提供 Web 字体无法加载时的备用方法。

幸好，第二种方法所需的 CSS 与第一种方法的差不多。因此，我们将在自己的样式表中使用 PT Sans stylesheet.css 中的第一个文件里的四个 `@font-face` 样式规则，并根据需要对其进行修改。现在，我们只需要用到字体的常规样式。下一节将用到字体的斜体、加粗以及加粗斜体等样式。

我们将使用图 13.5.1 中的 HTML，并将 CSS 放入 example.css（这个文件目前是空的）。尽管我们要使用的是 PT Sans，不过下面我们将按照一种通用的方式介绍各个步骤，从而可以很方便地将这些步骤应用到 Font Squirrel 中的任何字体。同样地，这种方法也适用于任何自托管的 Web 字体。

使用 Web 字体为普通文本设置字体样式的步骤

(1) 看看 13.3 节中介绍的从 Font Squirrel 下载的 ZIP 文件中的 web fonts 文件夹。如果你看到一个字体的常规文本版本的子文件夹，就打开该子文件中的 stylesheet.css。

(2) 将为常规文本准备的 `@font-face` 样式规则复制到你自己的样式表中（如图 13.5.2 所示）。

(3) 将字体文件（图 13.5.2 所示的样式表中引用的字体文件）从 Font Squirrel 文件夹复制到包含你的样式表的文件夹（如图 13.5.3 所示）。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Styling Text with a Web Font </title>
    <link rel="stylesheet"
    → href="example.css" />
</head>
<body>
<h1>Local Teen Prefers Vinyl Over Digital</h1>

<p>A local teenager has replaced all her
→ digital tracks with vinyl. "It's <em>
→ really</em> groovy," she said, on the
→ record. Without skipping a beat, she added,
→ "Besides, it's like going through a time
→ warp."</p>

<p>Some of her iPod-toting classmates aren't
→ as enthusiastic. "Yeah, they needle me
→ about it. What a bunch of ones and
→ zeros."</p>
</body>
</html>
```

图 13.5.1　我将从一个简单的 HTML 页面和一个空白的样式表开始。为了简化，两个文件都放在同一个目录。注意，该目录下并没有包含任何 PT Sans 字体文件

```
/* example.css */

@font-face {
    font-family: 'pt_sansregular';
    src: url('PTS55F-webfont.eot');
    src: url('PTS55F-webfont.eot?#iefix')
        → format('embedded-opentype'),
        url('PTS55F-webfont.woff')
        → format('woff'),
        url('PTS55F-webfont.ttf')
        → format('truetype');
    font-weight: normal;
    font-style: normal;
}
```

图 13.5.2　我更新了 example.css，从而让它包含 ptsans_regular_english 文件夹中 stylesheet.css 里的 `@font-face` 样式规则

(4) 对 `font-family` 值进行重命名，让该名称成为字体的代号。例如，使用字体的名

称、短横线和风格名称（如图 13.5.4 所示）。（你选择的名称并不重要，重要的是你需要在 font-family 中对该字体支持的所有风格（如常规、加粗、斜体、加粗斜体），都要使用**同样的**名称。当你通过下一节学会如何应用不同的风格以后，这样做会显得尤其重要。）

(5) 使用你最熟悉的方法，创建为文本元素添加样式的规则。为应用 Web 字体，在样式规则中输入 font-family: '*Web Font Name*';，其中的 *Web Font Name* 是第 (4) 步中创建的名称（如图 13.5.5 所示）。最后，一定要保存文件。

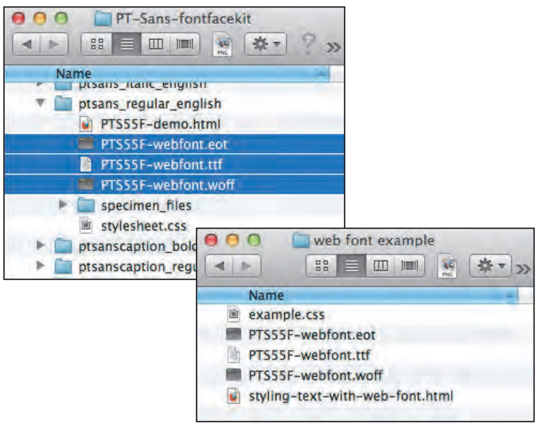


图 13.5.3　我将从 Font Squirrel 下载的文件夹中的三个 PT Sans Regular 字体格式文件（上）复制到我的示例文件夹（下）

```
/* example.css */

@font-face {
     font-family: 'PTSans';
     src: url('PTS55F-webfont.eot');
     ...
}
```

图 13.5.4　对 example.css 所做的唯一改动是将 pt_sansregular 替换为 PTSans，这样的名称更接近字体本身

```
/* example.css */

body {
     font-family: 'PTSans', sans-serif;
     font-size: 100%;
     line-height: 1.25;
}

@font-face {
     font-family: 'PTSans';
     src: url('PTS55F-webfont.eot');
     ...
}
```

图 13.5.5　这是对 body 应用的样式，看起来跟常规的 CSS 没有两样。字体栈中的 font-family 名（在这个例子中为 PTSans）必须与 @font-face 样式规则中的名称一致。这里的 font-size 和 line-height 并不是让 Web 字体生效的必要样式规则

如果你刷新页面，就会看到页面已经能显示 Web 字体（如图 13.5.6 所示）。不过，我们还没有结束。第一眼看上去好像没有问题，但仔细看看就会发现，页面上呈现的粗体和斜体是伪粗体（标题）和伪斜体（单词 really）。我将在下一步介绍如何修正这一问题。



图 13.5.6　由于为 body 应用了 PT Sans 样式，因此整个页面的文字用的都是这种字体

提示　通常，我会将图 13.5.5 中的第一条规则写为 `body { font: 100%/1.25 'PTSans', sans-serif; }`，即使用 `font` 简记法。这里将规则分开来写，是为了更清楚地演示应用 Web 字体的方法本身。

提示　考虑到简化，上述步骤和示例中均假定 HTML、样式表和字体文件均位于同一个目录，但在实践中最好将它们分开。13.6 节的最后一条提示介绍了一种方法。

使用单引号还是双引号

Font Squirrel 提供的 `@font-face` 样式规则使用单引号而非双引号包围字体名称。本书的 CSS 示例中也都是这样做的。在 CSS 里，单引号和双引号是完全一样的，因此你可以根据自己的喜好挑选任何一种方式。

`@font-face` 方法

Roger Johansson 在他的文章中总结了在样式表中使用 Web 字体的两种方法。文章的 URL 很长，不过访问 www.htmlcssvqs.com/8ed/font-face，页面会跳转到他的网站（www.456bereastreet.com）上的那篇文章。他首先介绍了 Font Squirrel 的方法，接着介绍了本章演示的方法。

我在前面提到过，我倾向于使用第二种方法。不过这种方法也有一定的局限性：如果对一种 Web 字体（并非页面上所有的 Web 字体）使用四种样式和粗细版本，Internet Explorer 6 ~ 8 会将特殊的样式和粗细版本直接以常规样式显示。

实践中，这对绝大多数情形都没有影响。首先，我们通常不会对同一种字体引入四种不同的样式和粗细版本。要知道，使用那么多样式和粗细版本，就需要浏览器下载比正常情况更多或更大的文件。

更重要的是，在这三个 IE 版本中，只有 IE8 在世界上大多数地区拥有一定的用户基数，而且其用户份额还在不断下降。例如，截至本书写作之际，IE8 在欧洲的份额为 6.9%，在北美为 9.5%。一年前，它的份额还有 40% 甚至更高。

13.6　为 Web 字体应用斜体和粗体

如果你想对 Web 字体添加一些基本样式，这时 Web 字体用起来有些奇怪。需要记住的是，对于 Web 字体，每个字体只有一种粗细和一种风格。如果你想使用粗体或斜体，就需要为它们创建单独的样式规则，每条规则对应一个 Web 字体文件。否则，浏览器就会使用伪粗体或伪斜体（如图 13.6.1 所示），或者同时使用伪粗体和伪斜体（后面将会看到）。



图 13.6.1　在前一节的例子中，浏览器让常规文本变得稍微粗一点来模拟粗体，让常规文本稍微倾斜来模拟斜体。然而，我们需要的是专门为该字体设计的加粗或斜体版本。关于伪斜体和伪粗体，请回顾 10.4 节和 10.5 节

在前一节中，我们没有在样式表中为任何文本指定加粗或斜体的样式（参见图 13.6.2）。那么这些样式是怎么产生的呢？我想你一定能猜出来，即浏览器的默认样式。浏览器会对 h1 应用粗体，对 em 应用斜体（如图 13.6.3 所示），就像你应用 Web 字体之前那样。

```
body {
    font-family: 'PTSans', sans-serif;
    font-size: 100%;
    line-height: 1.25;
}

@font-face {
    font-family: 'PTSans';
    src: url('PTS55F-webfont.eot');
    ...
    font-weight: normal;
    font-style: normal;
}
```

图 13.6.2　我们不会在 example.css 中的任何位置使用 font-weight: bold; 或者 font-weight: italic;

```
...
<h1>Local Teen Prefers Vinyl Over Digital
➝ </h1>

<p>..."It's <em>really</em> groovy," she said,
➝ on the record...</p>
...
```

图 13.6.3　我们的标题是 h1，我们希望强调的文本放在 em 中

幸好要修复这一问题相当简单。步骤与对常规文本实施 Web 字体是相似的，只是 CSS 部分有一些变动。跟之前一样，我会以 PT Sans 为例进行讲解，不过这些步骤也适用于从 Font Squirrel 获取到的其他字体。

1. 为 Web 字体应用斜体的步骤

(1) 看看 13.3 节中介绍的从 Font Squirrel 下载的 ZIP 文件中的 web fonts 文件夹。如果你看到一个字体的斜体版本的子文件夹，就打开该子文件中的 stylesheet.css。

(2) 将为常规文本准备的 @font-face 样式规则复制到你自己的样式表中（如图 13.6.4 所示）。

(3) 将斜体字体文件（图 13.6.4 所示的样式表中引用的字体文件）从 Font Squirrel 文件夹复制到包含你的样式表的文件夹（如图 13.6.5 所示）。

(4) 对 font-family 值进行重命名，让该名称与针对常规文本的样式规则中的名称保持一致（如图 13.6.6 所示）。

(5) 将斜体文本 @font-face 样式规则中的 font-style 值改为 font-style: italic（如图 13.6.6 所示）。最后，一定要保存文件。

```
body {
    font-family: 'PTSans', sans-serif;
    ...
}

/* 常规文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS55F-webfont.eot');
    ...
}

/* 斜体文本 */
@font-face {
    font-family: 'pt_sansitalic';
    src: url('PTS56F-webfont.eot');
    src: url('PTS56F-webfont.eot?#iefix')
        ➝ format('embedded-opentype'),
        url('PTS56F-webfont.woff')
        ➝ format('woff'),
        url('PTS56F-webfont.ttf')
        ➝ format('truetype');
    font-weight: normal;
    font-style: normal;
}
```

图 13.6.4　现在 example.css 包含了 PT Sans Italic 的 @font-face 规则

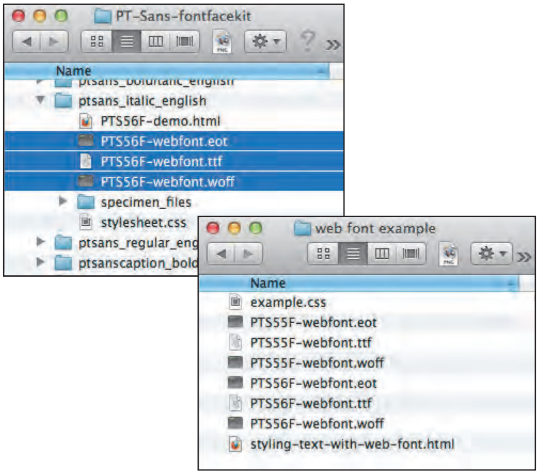


图 13.6.5　我将从 Font Squirrel 下载的文件夹中的三个 PT Sans Italic 字体格式文件（上）复制到我的示例文件夹（下）

```
body {
    font-family: 'PTSans', sans-serif;
    ...
}

/* 常规文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS55F-webfont.eot');
    ...
    font-weight: normal;
    font-style: normal;
}

/* 斜体文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS56F-webfont.eot');
    ...
    font-weight: normal;
    font-style: italic;
}
```

图 13.6.6　这里对代码做了两个重要的改动。将 `font-family: 'pt_sansitalic'` 改成了 `font-family: 'PTSans'`，从而与为常规文本 `@font-face` 样式规则中取的名称保持一致。然后在斜体文本 `@font-size` 样式规则中对 `font-style` 的值作了修改，以反映斜体样式

现在，只要 `font-style: italic;` 存在，无论是来自浏览器的默认样式（如图 13.6.7 所示）还是你自己的样式（如图 13.6.8 所示），都会显示斜体的 Web 字体。

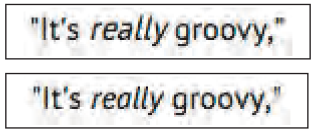


图 13.6.7　之前的伪斜体（上）与正确的版本（下）之间的差别在字体 a 和 e 上表现得最为明显。真正的斜体比伪斜体要小一些，与常规文本放在一起会显得更为协调

```
body {
    font-family: 'PTSans', sans-serif;
    ...
}

p {
    font-style: italic;
}

/* 常规文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS55F-webfont.eot');
    ...
    font-weight: normal;
    font-style: normal;
}

/* 斜体文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS56F-webfont.eot');
    ...
    font-weight: normal;
    font-style: italic;
}
```

图 13.6.8　出于演示需要，我将所有段落文本都以斜体显示。这一样式在接下来的示例中将被移除

2. 为 Web 字体应用粗体的步骤

(1) 重复“为 Web 字体应用斜体的步骤”中的第 (1) 步 ~ 第 (3) 步，但是要使用字体

的**粗体**版本对应的文件夹中的文件。将**粗体**文本的 @font-face 样式规则复制到样式表中（对应应用斜体样式步骤中的第 (2) 步，如图 13.6.9 所示），并将**粗体**字体文件复制到你的文件夹中（对应应用斜体样式步骤中的第 (3) 步，如图 13.6.10 所示）。

```
...

... 常规文本和斜体文本的@font-face规则 ...

/* 粗体文本 */
@font-face {
    font-family: 'pt_sansbold';
    src: url('PTS75F-webfont.eot');
    src: url('PTS75F-webfont.eot?#iefix')
        → format('embedded-opentype'),
        url('PTS75F-webfont.woff')
        → format('woff'),
        url('PTS75F-webfont.ttf')
        → format('truetype');
    font-weight: normal;
    font-style: normal;
}
```

图 13.6.9　现在 example.css 包含了 PT Sans Bold 的 @font-face 规则

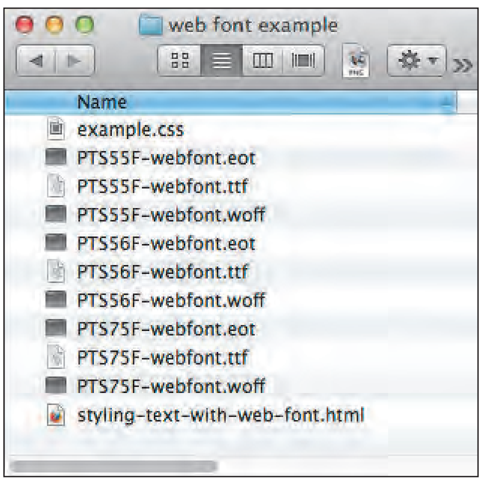


图 13.6.10　我将三个 PT Sans 粗体文件从 Font Squirrel 文件夹（未显示在图中）复制到示例文件夹。文件名均以 PTS75F 开头

(2) 对粗体文本 @font-face 样式规则中的 font-family 值进行重命名，使之与针对常规文本的样式规则中的名称保持一致（如图 13.6.11 所示）。

(3) 将粗体文本 @font-face 样式规则中的 font-weight 值改为 font-weight: bold（如图 13.6.11 所示）。最后，一定要保存文件。

```
body {
    font-family: 'PTSans', sans-serif;
    ...
}

/* 常规文本 */
@font-face {
    font-family: 'PTSans';
    ...
    font-weight: normal;
    font-style: normal;
}

/* 斜体文本 */
@font-face {
    font-family: 'PTSans';
    ...
    font-weight: normal;
    font-style: italic;
}

/* 粗体文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS75F-webfont.eot');
    ...
    font-weight: bold;
    font-style: normal;
}
```

图 13.6.11　我将 font-family: 'pt_sansbold' 改成了 font-family: 'PTSans'，从而与为常规文本 @font-face 样式规则中取的名称保持一致。然后在粗体文本 @font-size 样式规则中对 font-weight 的值作了修改，以反映粗体样式

现在，只要对文字设置加粗样式，无论是来自浏览器的默认样式（如图 13.6.12 中的 h1）还是你自己的样式，都会显示粗体的 Web 字体。例如，在图 13.6.11 中加入 p

{ font-weight: bold; }，就会让所有的段落文本变成加粗样式（需要注意的是，这也影响了 em 元素）。



图 13.6.12　如果对伪粗体（上）和正确的版本（下）进行比较，会发现正确的版本中字母更清晰，字间距更合理。现在，我们的标题为正确的粗体，em 文本为正确的斜体，其他文本均为常规样式

3. 为 Web 字体应用粗斜体的步骤

前面，我们已经看到如何为文本应用常规、加粗和斜体的 Web 字体。这也为我们留下了一个疑问：如何让使用 Web 字体的文本既有加粗样式也有斜体样式。

记住，每个 Web 字体都只对应一种样式、一种粗细版本。在我们引入的三种 PT Sans 字体中（参见图 13.6.11），没有一种是为 font-weight: bold 和 font-style: italic 同时存在而准备的。要么是其中的一种，要么都没有。这就是图 13.6.13 中的代码不会照你设想的方式运行的原因。（这解释了设置 p { font-weight: bold; } 后 em 会使用伪斜体的原因，因为 em 继承了 p 的加粗样式。）

下面来为页面添加粗斜体样式。

重复“为 Web 字体应用斜体的步骤”中的第 (1) 步 ~ 第 (3) 步，但是要使用字体的**粗斜体**版本对应的文件夹中的文件。将**粗斜体**文本的 @font-face 样式规则复制到样式表中（对应应用斜体样式步骤中的第 (2) 步，如图 13.6.14 所示），并将**粗斜体**字体文件复制到你的文件夹中（对应应用斜体样式步骤中的第 (3) 步，如图 13.6.15 所示）。

```
body {
    font-family: 'PTSans', sans-serif;
    ...
}

em {
    font-weight: bold;
}

... @font-face rules ...
```



图 13.6.13　我在图 13.6.11 中的样式表中添加了一条规则，尝试让 em 文本既是斜体（默认样式）又是粗体。但这里有个问题：我们没有加载 PT Sans 粗斜体字体，因此 em 有伪斜体样式（不过这有些难以观察）

```
...

em {
    font-weight: bold;
}

... 其他@font-face规则 ...

/* 粗斜体文本 */
@font-face {
    font-family: 'pt_sansbold_italic';
    src: url('PTS76F-webfont.eot');
    src: url('PTS76F-webfont.eot?#iefix')
       → format('embedded-opentype'),
       url('PTS76F-webfont.woff')
       → format('woff'),
       url('PTS76F-webfont.ttf')
       → format('truetype');
    font-weight: normal;
    font-style: normal;
}
```

图 13.6.14　使用图 13.6.13 的代码，现在 example.css 包含了为 PT Sans 粗斜体准备的 @font-face 样式规则

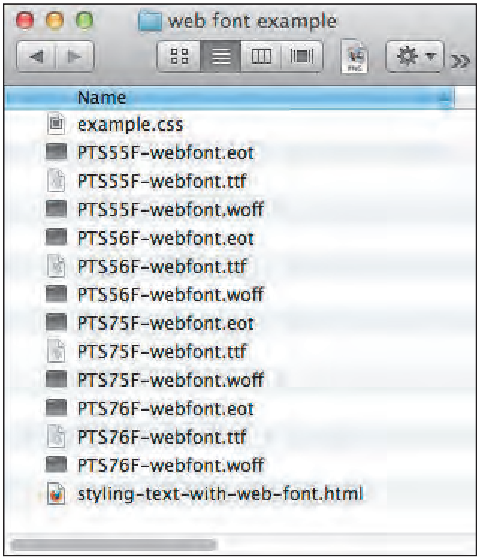


图 13.6.15　我将三个 PT Sans 粗斜体文件从 Font Squirrel 文件夹（未显示在图中）复制到示例文件夹。文件名均以 PTS76F 开头

对粗斜体文本 `@font-face` 样式规则中的 `font-family` 值进行重命名，使之与针对常规文本的样式规则中的名称保持一致（如图 13.6.16 所示）。

将粗斜体文本 `@font-face` 样式规则中的 `font-weight` 值改为 `font-weight: bold`，同时将 `font-style` 值改为 `font-style: italic`（如图 13.6.16 所示）。最后，一定要保存文件（如图 13.6.17 所示）。

提 示　如果某种 Web 字体没有粗体、斜体或粗斜体版本，而我们又对文本添加了这些样式，浏览器就会显示伪样式。

提 示　有些字体（如 ChunkFive）本身就很粗，没有粗体版本。应该对这类字体使用 `font-weight: normal;`（即其常规状态），因为 `font-weight: bold;` 会让浏览器将其显示伪粗体，这样它们的笔划就更粗了。

```
body {
    font-family: 'PTSans', sans-serif;
    ...
}

em {
    font-weight: bold;
}

/* 常规文本 */
@font-face {
    font-family: 'PTSans';
    ...
    font-weight: normal;
    font-style: normal;
}

/* 斜体文本 */
@font-face {
    font-family: 'PTSans';
    ...
    font-weight: normal;
    font-style: italic;
}

/* 粗体文本 */
@font-face {
    font-family: 'PTSans';
    ...
    font-weight: bold;
    font-style: normal;
}

/* 粗斜体文本 */
@font-face {
    font-family: 'PTSans';
    src: url('PTS76F-webfont.eot');
    ...
    font-weight: bold;
    font-style: italic;
}
```

图 13.6.16　我将 `font-family: 'pt_sansbold_italic'` 改成了 `font-family: 'PTSans'`，从而与为常规文本 `@font-face` 样式规则中取的名称保持一致。然后对 `font-weight` 和 `font-style` 的值都作了修改，以反映粗斜体样式

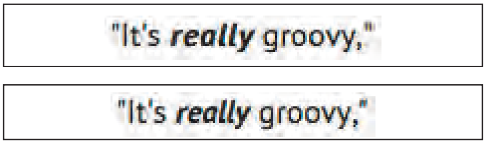


图 13.6.17 变化很微妙，伪效果（上）与正确的字体（下）相比，字母更粗一些，字间距也更大一些

提 示 记住，每一个样式和粗细版本都会引入一个新的字体文件，从而增加浏览器要下载的文件大小，这会影响性能。出于这个原因，很多设计师仅对标题使用 Web 字体。

提 示 由于引入了大量的文件，我们的文件夹看起来有些乱（如图 13.6.15 所示）。现在你应该能看出将字体文件与样式表放在不同文件夹的原因了！图 13.6.18 显示了一种组织这些文件的方式。你可能需要根据这种结构调整样式表中的每个 `url`。例如，`url('font/PTS55Fwebfont.woff')`。同时，你还需要修改 HTML 中引入样式表的 `link` 元素以反映正确的位置：`<link rel="stylesheet" href="css/example.css" />`。

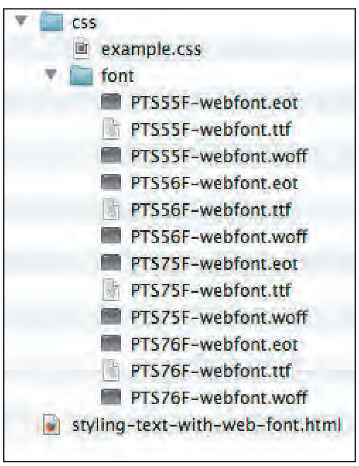


图 13.6.18 在这种结构中，字体文件都位于 font 文件夹，该文件夹又位于 css 文件夹中

13.7 使用 Google Fonts 的 Web 字体

前面的小节介绍了如何将自托管的 Web 字体添加到页面中去，例如使用 Font Squirrel 提供的字体。在众多托管 Web 字体的服务中，我们来看看 Google Fonts（参见 13.1 节）。

同其他字体服务一样，Google Fonts 也为我们省下了创建和修改 `@font-face` 规则的麻烦。这是自托管字体和使用 Google Fonts 这类服务之间的一个明显的区别（也可以说是 Google Fonts 这类服务的一个优势）。

不像其他的服务，Google Fonts 是免费的，它有几百种字体可供选择，而且使用它在页面中添加 Web 字体不到一分钟就能搞定，因此它已经变得越来越流行。一旦选中一种字体，省下我们要做的全部事情就是将一段 Google Fonts 代码添加到 HTML 页面中，然后就可以在 CSS 中为文本应用字体样式了。

使用 Google Fonts 的 Web 字体的步骤

(1) 访问 www.google.com/fonts。找到你喜欢的字体后，点击 Add to Collection（添加到集合），如图 13.7.1 所示。

(2) 将页面滚动到底部，点击 Use（使用），如图 13.7.2 所示。

(3) 仅选择将要使用的字体样式和粗细版本。这样做能让字体文件尽可能小（如图 13.7.3 所示）。



图 13.7.1 这里将 Lato 添加到了我的 Web 字体集合。此外还添加了 Open Sans（不过这里没有显示）。这是我为第 11 章和第 12 章中的大部分示例页面中的标题使用的字体



图 13.7.2　页面的底部列出了你选择的字体（上），这块区域还有三个按钮（下）。Choose（选择）按钮对应你现在所在的状态——浏览字体，选择喜欢的字体。点击 Review（回顾）按钮可以看到更多的例子，以及集合中字体的更多信息。点击 Use 按钮，便可以在页面中使用所选的字体

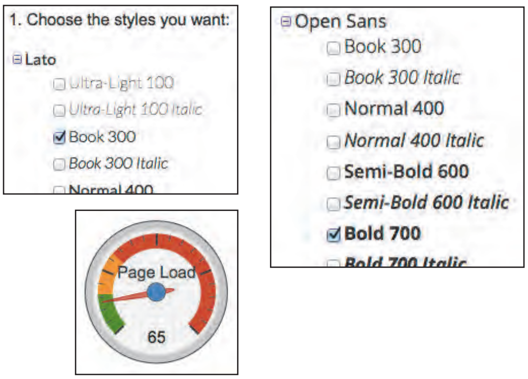


图 13.7.3　我的页面对每种字体均只需要一种粗细版本。Page Load（页面加载）仪表指示了字体文件大小对使用这些字体的页面的加载速度的影响。现在看起来还不错，不过如果添加了其他的粗细版本，页面的加载时间就会变长

(4) 仅选择内容所需的字符（如图 13.7.4 所示），这也会影响文件的大小。

(5) 将 link 元素代码（如图 13.7.5 所示）复制到你的网页的 head 中去（如图 13.7.6 所示）。

(6) 在 font-family 中使用 Google Fonts 提供的名称，便可以对文本应用字体样式。对 font-weight 使用第 (3) 步选择的一种粗细版本（如图 13.7.7 所示）。如果使用了一种斜体的 Web 字体，则在 CSS 样式规则中设置 @font-style: italic;。

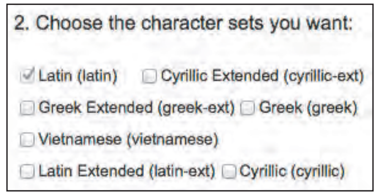


图 13.7.4　你的内容所需的字符对应的字体子集。默认为 Latin

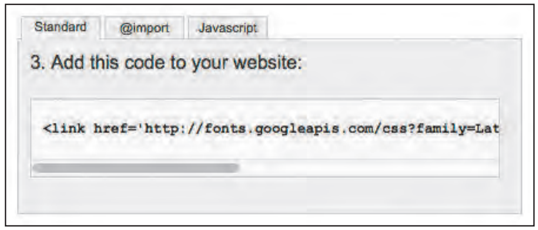


图 13.7.5　复制 Standard（标准）标签下的 link 代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Le Journal</title>
    <link href="http://fonts.googleapis.
    → com/css?family=Lato:300|Open+
    → Sans:700" rel="stylesheet" />
    <link rel="stylesheet" href="style.css" />
</head>
<body>
…
<h1>Sunny East Garden at the Getty Villa</h1>

<p>It is hard to believe, but there are
→ about...</p>

…

<h2>Popular Posts</h2>
…
</body>
</html>
```

图 13.7.6　将 Google Fonts 提供的 link 代码粘贴到自己的样式表（style.css）的 link 的前面。注意，Google Fonts 提供的 link 中的 href 值包含了我之前选择的字体名称和粗细版本（参见图 13.7.2 和图 13.7.3）。这样就会加载所有的 Web 字体文件以及使用它们所需的 @font-face 样式规则

(7) 保存 HTML 页面和样式表。

在浏览器中查看页面（如图 13.7.8 所示），可以看到 Web 字体已经生效了。

```
body {
    font: 100%/1.2 Georgia, 'Times New Roman',
serif;
}

h1 {
    color: #333;
    font-family: 'Lato', sans-serif;
    font-size: 3.25em; /* 52px/16px */
    font-weight: 300;
    letter-spacing: -2px;
    line-height: .975;
    margin-bottom: .4125em;
}

h2 {
    border-bottom: 1px solid #dbdbdb;
    font-family: 'Open Sans',
sans-serif;
    font-size: .875em; /* 15px/16px */
    font-weight: 700;
    text-transform: uppercase;
    padding-bottom: .75em;
}

... 本页面的更多CSS ...
```

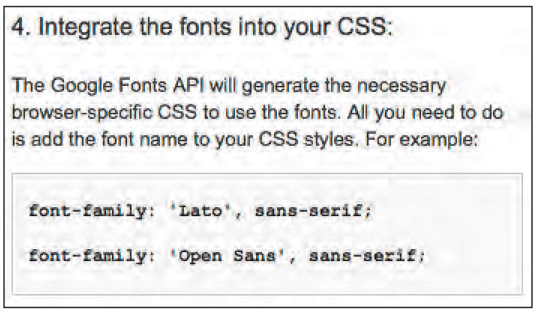


图 13.7.7　Google Fonts（下）显示了引用字体的方法。我对 h1 元素应用了粗细为 300 的 Lato 字体，对 h2 元素应用了粗细为 700 的 Open Sans 字体。这些 font-weight 数值与 link 元素中的数字（参见图 13.7.6）是对应的，而 link 元素中的数字则与我所选的粗细版本（参见图 13.7.3）对应。这里显示的其他 CSS 并不是让 Web 字体生效的必要代码



图 13.7.8　大标题是 Lato 字体，小一些的粗体则是 Open Sans 字体

提 示　如果你想对斜体文本应用字体样式，一定要确保在选择 Web 字体样式和粗细版本时（第 (3) 步）选择斜体版本，并在定义 CSS 时（第 (6) 步）指定了正确的 font-weight 值以及 font-style: italic。否则就会显示伪斜体。类似地，如果不引入字体的粗体版本，浏览器就会显示伪粗体。更多示例与讨论参见 13.6 节以及 10.4 节和 10.5 节。

第 14 章

使用 CSS3 进行增强

14

本章内容

- ❑ 浏览器兼容性、渐进增强和 polyfill
- ❑ 理解厂商前缀
- ❑ 为元素创建圆角
- ❑ 为文本添加阴影
- ❑ 为其他元素添加阴影
- ❑ 应用多重背景
- ❑ 使用渐变背景
- ❑ 为元素设置不透明度
- ❑ 生成内容的效果
- ❑ 使用 sprite 拼合图像

网站制作者多年来面临的挑战之一就是，使用 CSS 建立丰富布局的选择是很有限的。在大多数情况下，要建立丰富的布局，就需要使用额外的 HTML、CSS 及大量图像。这样做的结果就是，页面变得更为复杂，可访问性降低，浏览器需要花更长的时间下载和显示页面，同时，页面变得更为脆弱，更难维护。

近年来，浏览器快速吸纳了很多新的 CSS3 属性，让上述情况有了改观。如今，仅使用 CSS 创建圆角、渐变和阴影以及调整透明度等已经变成现实。你在本章将看到如何实现这些效果，以及如何应对不支持这些特性的旧浏览器。

这样做的结果是网页可以使用更少的标记和图像，这样加载的速度也会变快。这对所有的用户来说都有好处，尤其是对使用处理能力较低的设备（如智能手机）的用户。

14.1　浏览器兼容性、渐进增强和 polyfill

由于浏览器发展的脚步在近几年明显地加快了，因此理解这些新的 CSS 属性在什么时候会得到预期的效果就很重要了。这里给出了浏览器对本章将要介绍到的属性开始提供基本支持的时间，如图 14.1.1 所示。

border-radius	3.0	9.0	1.0	3.1	10.5
box-shadow	3.5	9.0	1.0	3.1	10.5
text-shadow	3.5	10.0	1.0	3.1	9.5
多重背景	3.6	9.0	1.0	3.1	10.5
渐变	3.6	10.0	2.0	4.0	11.1
透明度	1.0	9.0	1.0	3.1	9.0
before 和 :after	1.0	8.0	1.0	3.1	9.0

图 14.1.1　这个表格仅列出了桌面浏览器，不过也不要忽视移动浏览器，移动浏览器的份额每天都在快速地增长。关于移动浏览器的信息，以及桌面浏览器的更多信息，参见 Alexis Deveria 的 Can I Use 网站（www.caniuse.com），该表中的大部分信息都来自这个网站。还有一份更为简洁的支持性总结位于 http://fmbip.com/litmus/

这里列出的大部分浏览器版本都有些年代了，只有 IE8 仍然占据了较高的份额。截至本书写作之际，IE8 在全球的市场份额大约为 8.5%，并处于不断下滑的态势。IE8 的用户无法体验本章列举的大部分特性，不过你将看到，这没关系。

1. 渐进增强

你在前面已经多次听我提到渐进增强了，首次提到这个词是在本书的前言。简单说来，它强调创建所有用户都能访问（无论使用什么样的 Web 浏览器）的基本层面的内容和功能，同时为更强大的浏览器提供增强的体验。更简单一些，渐进增强意味着网站在不同 Web 浏览器中的外观和行为不一样是完全可以接受的，只要内容是可访问的。

Dribbble 网站（http://dribbble.com）就是渐进增强的一个实例，如图 14.1.2 所示。通过渐进增强，该网站使用 CSS3 为现代浏览器提供更丰富的体验。在旧的浏览器（如 Internet Explorer 8）中，该网站呈现出稍微不同的视觉体验，但功能并未削减，如图 14.1.3 所示。



图 14.1.2　Dribbble 网站使用了一些 CSS3 属性（如 `border-radius` 和 `box-shadow`）为使用现代浏览器的用户提供更为丰富的体验，同时，它也记得为功能较弱的浏览器着想，参见图 14.1.3

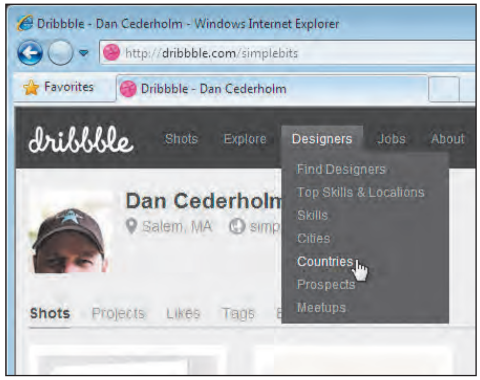


图 14.1.3　在不支持 `border-radius` 或 `box-shadow` 的旧浏览器（如 Internet Explorer 8）中查看该网站，体验就会发生改变。下拉菜单中各项的渐阴影不见了，圆角变成直角了，但功能并未损失，一切照常工作。这是渐进增强在实践中的一个方面

本章稍后为大家介绍如何对 About Me 应用渐变和阴影，如图 14.1.4 所示。IE8 显示的是没有阴影的纯色背景，但内容依旧清晰易读。



图 14.1.4　左图为现代浏览器中显示的效果，右图为 IE8 及其他旧浏览器中呈现的简化版本

2. 为不支持某些属性的浏览器使用 polyfill

如果你想弥合较弱的浏览器与较强的浏览器之间的功能差异，可以使用 polyfill（通常又称**垫片**，shim）。

polyfill 通常使用 JavaScript 实现，它可以为较弱的浏览器提供一定程度的对 HTML5 和 CSS3 的 API 和属性的支持，同时，当浏览

器本身就具有相应的能力时，会不动声色地退而使用官方的支持。需要注意的是，这样做通常会对性能产生一定的负面影响，因为较弱的 Web 浏览器（尤其是旧版本的 Internet Explorer）运行 JavaScript 的速度要慢得多。

关于 IE，Jason Johnston 的 CSS3 PIE（http://css3pie.com）是一种最为常见的 polyfill[①]。它为 Internet Explorer 6 ~ 9 提供了本章讨论的大部分 CSS 效果的支持。（其中，IE9 仅需要 PIE 提供对线性渐变的支持；对于其他的效果，IE9 提供原生的支持。）

提示　HTML5 Please（http://html5please.com）是一个不错的资源，使用它可以找出哪些 HTML5 和 CSS3 可以放心使用，以及有哪些好的填补差距的 polyfill。

提示　Modernizr（www.modernizr.com）是一个 JavaScript 库，它允许你探测浏览器是否支持创建优化的网站体验所需的特定的 HTML5、CSS3 及其他的特性。

更多的 CSS3 效果

相比这些页面中应用的效果，CSS3 能够实现的效果要多很多，如变换、过渡、动画等类别。例如，我们可以创建滑过链接时渐变的效果：

```
a { /* 简单起见，省略了前缀 */
    color: #007c21;
    transition: color .4s ease;
}
a:hover { color: #00bf32; }
```

要想找到学习更多内容的链接，参见 www.htmlcssvqs.com/resources/。

14.2　理解厂商前缀

CSS3 规范要达到 W3C 的推荐标准（即定稿）状态要经过数年。浏览器则通常在 W3C 开发标准的过程中就会提前实现这些特性。这样，标准在最终敲定之前就能知道哪些地方还能进一步改进。

在包含某个特性的初始阶段，浏览器通常会使用厂商前缀实现这类特性，参见图 14.2.1。这样，每个浏览器都可以引入自己的 CSS 属性支持方式，从而可以获取反馈，而且一旦标准发生改变也不会造成影响。

```
div {
    -webkit-border-radius: 10px;
    border-radius: 10px;
}
```

图 14.2.1　使用 border-radius 属性的示例。该属性需要使用 -webkit 前缀支持旧版本的 Android、iOS 和 Safari 浏览器。这些浏览器的一些新版本已经不需要使用带前缀的属性了，而只需使用无前缀的属性就行了（简单写成 border-radius: 10px;）。照例，样式规则中的最后一条声明优先级最高，这也是将无前缀版本放在最后的原因

不过，这种方式会造成一定程度的混乱，因此很多浏览器都逐渐取消了厂商前缀的使用。不过，正如你在本章看到的，还有一些地方需要使用它们，至少在依赖它们的旧浏览器依然占据一定市场份额的时候是这样。

每个主流浏览器都有其自身的前缀：-webkit-（Webkit/Safari/ 旧版本的 Chrome）、-moz-（Firefox）、-ms-（Internet Explorer）、-o-（Opera）。应该将前缀放在 CSS 属性名的前面，参见图 14.2.1。如今，在多数情况下，我们一般只需要 -webkit 前缀。

① polyfill 用于填补旧浏览器与新的 Web 技术之间的差异。——译者注

提示　并非所有的 CSS3 属性都需要使用为浏览器准备的前缀，如 text-shadow 和 opacity（本章稍后就会介绍）。

提示　网上有一些创建 CSS3 代码（包括厂商前缀）的工具，参见图 14.2.2。CSS3, Please! 列出了哪些浏览器支持无前缀的语法，哪些需要前缀。这对决定是否使用前缀这一问题来说是非常有用的——你可以自行决定一部分浏览器不显示 CSS3 效果是不是可接受的。



图 14.2.2　Randy Jensen 的 CSS3 Generator（www.css3generator.com），以及 Paul Irish 和 Jonathan Neal 的 CSS3, Please!（css3please.com）可以让你避免书写带前缀和不带前缀的 CSS 属性这种重复劳动。像 LESS、Sass 和 Stylus 这样的 CSS 预处理器处理这个问题会更方便。关于其他厂商前缀的信息，参见 http://css-tricks.com/how-to-dealwith-vendor-prefixes/

14.3　为元素创建圆角

使用 CSS3，可以在不引入额外的标记或图像的情况下，为大多数元素（包括表单元素、图像，甚至段落文本）创建圆角，如图 14.3.1 ~ 图 14.3.5 所示。

```
…
<body>
<div class="all-corners"></div>
<div class="one-corner"></div>
<div class="elliptical-corners"></div>
<div class="circle"></div>
</body>
</html>
```

图 14.3.1　这个文档包含带 class 属性的示例 div。每个 div 均用于演示 border-radius 的不同语法，包括创建相同的四个圆角，使用长形式语法创建单独的圆角，创建椭圆形圆角以及创建圆形等图形

```
div {
     background: #999;
     float: left;
     height: 150px;
     margin: 10px;
     width: 150px;
}

.all-corners {
     -webkit-border-radius: 20px;
     border-radius: 20px;
}

.one-corner {
     -webkit-border-top-left-radius:
75px;
     border-top-left-radius: 75px;
}

.elliptical-corners {
     -webkit-border-radius: 50px / 20px;
     border-radius: 50px / 20px;
}

.circle {
     -webkit-border-radius: 50%;
     border-radius: 50%;
}
```

图 14.3.2　使用 CSS 圆角的四个例子，包含了必要的厂商前缀以支持旧版 Android、Mobile Safari 和 Safari 浏览器。对于 .circle，使用 75px 与 50% 的效果是一样的，因为该元素的大小为 150 像素 × 150 像素

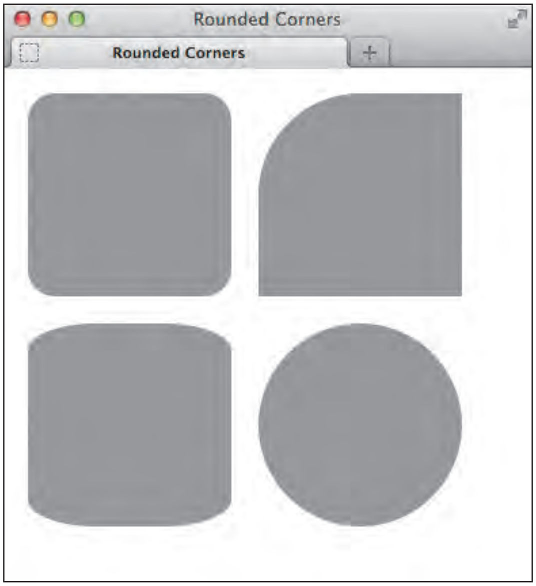


图 14.3.3　支持 border-radius 属性（包括有前缀的和无前缀的）的浏览器会以图中所示的方式将示例呈现出来。注意，不同的实现之间可能存在视觉上的差异，尤其是在旧版本的 Safari 和 Firefox 中，差异更为明显

```
.about {
    background-color: #2b2b2b;
    border-radius: 10px;
    padding: .3125em .625em .625em;
}

.about img {
    border: 5px solid #bebebe;
    border-radius: 15px;
}
```

图 14.3.4　这是一个真实的例子，对本书之前创建的我们熟悉的 About Me 模板应用圆角样式（如图 14.3.5 所示）。注意图像的样式规则中组合使用了 border 和 border-radius

同 border、margin 和 padding 属性一样，border-radius 属性也有长短两种形式的语法，参见图 14.3.6 和图 14.3.7。只有当你想在旧版本的 Android、Mobile Safari 和 Safari 浏览器中显示相应效果的时候才使用 -webkit 前缀。



图 14.3.5　外围的圆角比图像的 15 像素圆角半径更小一些。在旧浏览器中，这两个地方都显示为直角，就像第 11 章和第 12 章中显示的那样

```
div {
    background: #ff6;
    border: 5px solid #326795;
    ...
}

.example-1 {
    /* 将左上方和右下方圆角半径设为10px，右
       上方和左下方圆角半径设为20px */
    border-radius: 10px 20px;
}

.example-2 {
    /* 将左上方圆角半径设为20px，其他圆角半
       径设为0 */
    border-radius: 20px 0 0;
}

.example-3 {
    /* 将左上方圆角半径设为10px，右上方圆角
       半径设为20px，右下方圆角半径设为0，左
       下方圆角半径设为30px */
    border-radius: 10px 20px 0 30px;
}
```

图 14.3.6　更多使用简记法设置一个或多个圆角的例子，使用简记法就不必书写 border-top-left-radius 这样的属性了。简短起见，这里省略了 -webkit- 前缀

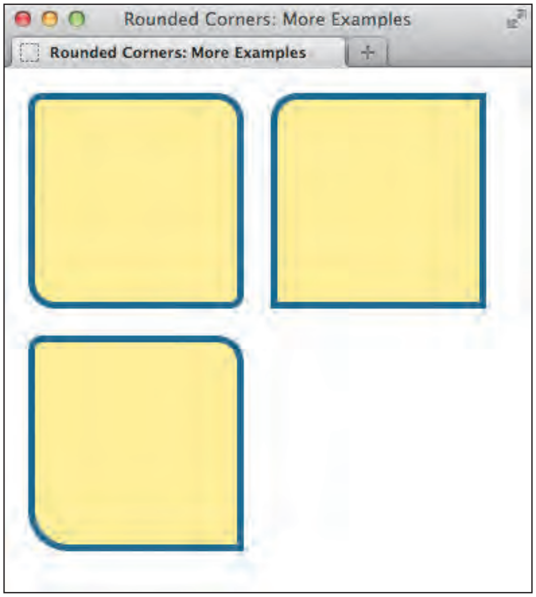


图 14.3.7　使用长语法设置的圆角样式

1. 为元素创建四个相同的圆角

(1) 这一步可选，输入 -webkit-border-radius: *r*，这里的 *r* 是圆角的半径大小，表示为长度（带单位）。

(2) 输入 border-radius: *r*，这里的 *r* 是圆角的半径大小，使用与第 (1) 步中相同的值，参见图 14.3.2。这是该属性的标准短形式语法。

2. 为元素创建一个圆角

(1) 这一步可选，输入 -webkit-border-top-left-radius: *r*，这里的 *r* 是左上方圆角的半径大小，表示为长度（带单位）。

(2) 输入 border-top-left-radius: *r*，这里的 *r* 使用与第 (1) 步中相同的值，参见图 14.3.2。这是该属性的标准长形式语法。

注意这些步骤仅演示了如何创建左上方圆角，此外，还可以单独创建其他方位的圆角，步骤如下。

- 创建右上方圆角：用 top-right 替换第 (1) 步和第 (2) 步中的 top-left。
- 创建右下方圆角：用 bottom-right 替换第 (1) 步和第 (2) 步中的 top-left。
- 创建左下方圆角：用 bottom-left 替换第 (1) 步和第 (2) 步中的 top-left。

3. 创建椭圆形圆角

(1) 这一步可选，输入 -webkit-border-radius: *x*/*y*，其中 *x* 是圆角在水平方向上的半径大小，*y* 是圆角在垂直方向上的半径大小，均表示为长度（带单位）。

(2) 输入 border-radius: *x*/*y*，其中 *x* 和 *y* 跟第 (1) 步中的值相等，参见图 14.3.2。

4. 使用 border-radius 创建圆形

(1) 输入 -webkit-border-radius: *r*，这里的 *r* 是元素的半径大小（带长度单位）。要创建圆形，可以使用短形式的语法，*r* 的值应该等于元素高度或宽度的一半。

(2) 输入 border-radius: *r*，这里的 *r* 是元素的半径大小（带长度单位），跟第 (1) 步中的 *r* 相等，参见图 14.3.3。这是标准的无前缀语法。

提 示　不支持 border-radius 的旧的浏览器仅会以方角呈现元素。

提 示　border-radius 仅影响施加该样式的元素的角，不会影响其子元素的角。因此，如果一个子元素有背景，该背景就有可能显示在一个或多个父元素的角的位置，从而影响圆角样式。

提 示　有时元素的背景（这里讲的不是子元素的背景）会透过其圆角。为了避免这种情况，可以在元素的 border-radius 声明后面增加一条样式规则：background-clip: padding-box;。

提示　图 14.3.4、图 14.3.5 和图 14.3.7 展示了如何结合使用 border 和 border-radius。

提示　border-radius 属性不是继承的。

14.4　为文本添加阴影

最早，text-shadow 是 CSS2 规范的一部分，接着在 CSS2.1 中被移除了，后来在 CSS3 中又出现了。使用该元素，可以在不使用图像表示文本的情况下，为段落、标题等元素中的文本添加动态的阴影效果（参见图 14.4.1 ~ 图 14.4.4）。

```
...
<body>

<p class="basic">Basic Shadow</p>
<p class="basic-negative">Basic Shadow</p>
<p class="blur">Blur Radius</p>
<p class="blur-inversed">Blur Radius</p>
<p class="multiple">Multiple Text Shadows</p>

</body>
</html>
```

图 14.4.1　演示 text-shadow 用法的段落名和类名

1. 为元素的文本添加阴影

(1) 输入 text-shadow:。

(2) 分别输入表示 x-offset（水平偏移量）、y-offset（垂直偏移量）、blur-radius（模糊半径）和 color 的值（前三个值带长度单位，四个值之间不用逗号分隔），例如 -2px 3px 7px #999（关于允许输入的值，参见提示）。

2. 为元素的文本添加多重阴影

(1) 输入 text-shadow:。

(2) 分别输入 x-offset、y-offset、blur-radius 和 color 的值（前三个带长度单位，四个值之间不用逗号分隔）。blur-radius 的值是可选的。

(3) 输入 ,（逗号）。

(4) 对四种属性使用不同的值重复第 (2) 步。

```
p {
    color: #222; /* 接近黑色 */
    font-size: 4.5em;
    font-weight: bold;
}

.basic {
    text-shadow: 3px 3px #aaa;
}

.basic-negative { /* 负偏移值 */
    text-shadow: -4px -2px #ccc;
}

.blur {
    text-shadow: 2px 2px 10px grey;
}

.blur-inversed {
    color: white;
    text-shadow: 2px 2px 10px #000;
}

.multiple {
    text-shadow:
      2px 2px white,
      6px 6px rgba(50,50,50,.25);
}
```

图 14.4.2　这些类演示了几种 text-shadow 的效果。第一个、第二个和第五个都省略了模糊半径的值。.multiple 类告诉我们，可以为单个元素添加多个阴影样式，每组属性之间用逗号分隔。这样，通过结合使用多个阴影样式，可以创建出特殊而有趣的效果

3. 将 text-shadow 改回默认值

输入 text-shadow: none;。

提示　text-shadow 属性不需要使用厂商前缀。

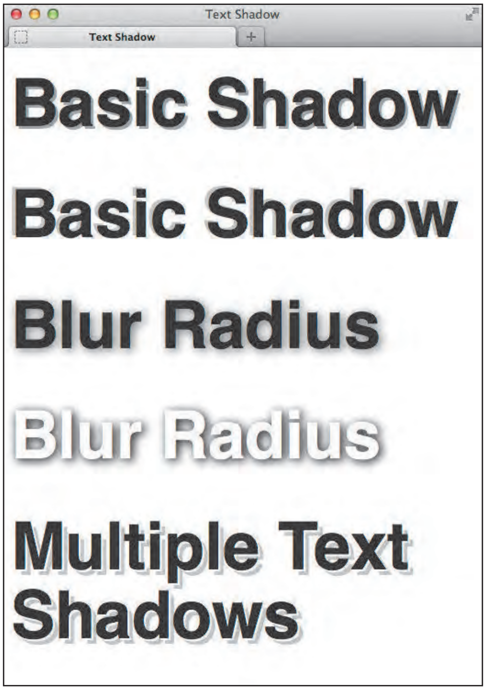


图 14.4.3 第一个段落有正的偏移量，第二个有负的偏移量。（偏移量的两个值并不需要同时为正或同时为负。）这两个段落都没有模糊半径，但第三个和第四个有。第四个有反色的效果，这是将文本颜色设为白色的结果（参见图 14.4.2）。最后一个有两个文本阴影，不过你还可以添加更多的阴影

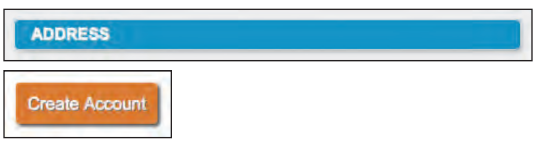


图 14.4.4 使用了 text-shadow 和 box-shadow（将在下一节介绍）的标题和第 16 章将要介绍的表单按钮。代码位于 www.htmlcssvqs.com/8ed/16

提 示 text-shadow 属性接受四个值：带长度单位的 x-offset、带长度单位的 y-offset、可选的带长度单位的 blur-radius 以及 color 值。如果不指定 blur-radius，将假定其值为 0（图 14.4.2 中有三个这样的例子）。

提 示 x-offset 和 y-offset 值可以是正整数，也可以是负整数，也就是说，1px 和 -1px 都是有效的。blur-radius 值必须是正整数。这三个值都可以为 0。

提 示 颜色可以表示为十六进制数、RGB、RGBA 或 HSLA 值（参见 7.5 节的“CSS 颜色”）。

提 示 尽管 text-shadow 的语法与边框和背景属性的语法是类似的，但它不能像边框和背景那样单独地指定四个属性值。

提 示 如果不对 text-shadow 的值进行设置，它就会使用初始值 none。

提 示 text-shadow 属性是继承的。

14.5 为其他元素添加阴影

使用 text-shadow 属性可以为元素的文本添加阴影，使用 box-shadow 属性则可以为元素本身添加阴影，参见图 14.5.1 ~ 图 14.5.6 所示。它们的基础属性集是相同的，不过 box-shadow 还允许使用两个可选的属性——inset 关键字属性和 spread 属性（用于扩张或收缩阴影）。

box-shadow 属性与 text-shadow 属性的另一个区别是，如果你希望兼容旧版 Android、Mobile Safari 和 Safari 浏览器，那么 box-shadow 需要加上 -webkit- 厂商前缀。关于这方面内容的最新信息参见 http://caniuse.com/#search=box-shadow。开发人员可以自行决定是否省略该前缀。

box-shadow 属性接受六个值：带长度单位的 x-offset 和 y-offset、可选的带长度单位

位的 blur-radius、可选的 inset 关键字、可选的带长度单位的 spread 值及 color 值。如果不指定 blur-radius 和 spread 的值，则设为 0。

```
...
<body>
<div class="shadow">
    <p>Shadow with Blur</p>
</div>

<div class="shadow-negative">
    <p>Shadow with Negative Offsets and
    ➝ Blur</p>
</div>

<div class="shadow-spread">
    <p>Shadow with Blur and Spread</p>
</div>

<div class="shadow-offsets-0">
    <p>Shadow with Offsets Zero, Blur, and
    ➝ Spread</p>
</div>

<div class="inset-shadow">
    <p>Inset Shadow</p>
</div>

<div class="multiple">
    <p>Multiple Shadows</p>
</div>
</body>
</html>
```

图 14.5.1　这个文档包含六个 div，它们用于演示使用 box-shadow 添加一个或多个阴影的效果

```
div {
    background: #fff;
    ...
}

.shadow {
    -webkit-box-shadow: 4px 4px 5px #999;
    box-shadow: 4px 4px 5px #999;
}

.shadow-negative {
    -webkit-box-shadow: -4px -4px 5px
    ➝ #999;
    box-shadow: -4px -4px 5px #999;
}

.shadow-spread {
    -webkit-box-shadow: 4px 4px 5px 3px
    ➝ #999;
    box-shadow: 4px 4px 5px 3px #999;
}

.shadow-offsets-0 {
    -webkit-box-shadow: 0 0 9px 3px #999;
    box-shadow: 0 0 9px 3px #999;
}

.inset-shadow {
    -webkit-box-shadow: 2px 2px 10px #666
    ➝ inset;
    box-shadow: 2px 2px 10px #666
inset;
}

.multiple {
    -webkit-box-shadow:
       2px 2px 10px rgba(255,0,0,.75),
       5px 5px 20px blue;

    box-shadow:
       2px 2px 10px rgba(255,0,0,.75),
       5px 5px 20px blue;
}
```

图 14.5.2　前五个类各自应用了一个彼此不同的阴影样式。最后一个类应用了两个阴影（还可以应用更多个阴影）。不理解 box-shadow 的浏览器会直接忽略这些 CSS 样式规则，呈现没有阴影的页面

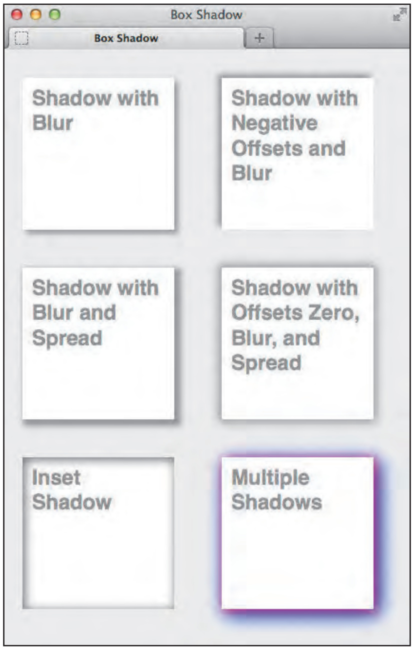


图 14.5.3　六个示例在支持 box-shadow 属性的浏览器中显示的样子



图 14.5.4　这个元素的样式为 box-shadow: 0 20px 10px -11px #999;。负的 spread 值会让阴影在元素内进行收缩。为 0 的 x-offset 值意味着阴影不会向左或向右偏离该元素

```
.about {
     border: 1px solid #d0d0d0;
     -webkit-box-shadow: 0 0 3px #d0d0d0;
     box-shadow: 0 0 3px #d0d0d0;
     padding: 0.313em 0.625em 0.625em;
}
```

图 14.5.5　这是一个真实的例子，即第 11 章和第 12 章的 About Me 模块。我将两个偏移量都设为 0，因此两个微小的阴影会出现在四个边上



图 14.5.6　About Me 模块有了一种新的样式。这时，它有一个 1 像素的边框，四个边上都有一个微小的阴影（深色的背景色被默认的白色取代了）

1. 为元素添加阴影

(1) 输入 -webkit-box-shadow:。

(2) 分别输入表示 x-offset、y-offset、blur-radius、spread 和 color 的值（前四个值均带长度单位），例如 2px 2px 3px 5px #333。

(3) 输入 box-shadow:，再重复第 (2) 步。

2. 创建内阴影

(1) 输入 -webkit-box-shadow:。

(2) 分别输入表示 x-offset、y-offset、blur-radius、spread 和 color 的值（前四个值均带长度单位），例如 2px 2px 5px #333。

(3) 在冒号后输入 inset，再输入一个空格（也可以在第 (2) 步之前输入 inset 和一个空格）。

(4) 输入 box-shadow:，再重复第 (2) 步和第 (3) 步。

3. 为元素应用多重阴影

(1) 输入 -webkit-box-shadow:。

(2) 分别输入表示 x-offset、y-offset、blur-radius、spread 和 color 的值（前四个值均带长度单位），例如 2px 2px 5px #333。如果有必要可以将 inset 关键字包含在内。

(3) 输入 ,（逗号）。

(4) 对每种属性使用不同的值重复第 (2) 步。

(5) 输入 box-shadow:，再重复第 (2) 步至第 (4) 步。

4. 将 box-shadow 改回默认值

(1) 输入 -webkit-box-shadow: none;。

(2) 输入 box-shadow: none;。

提示　x-offset、y-offset 和 spread 值可以是正整数，也可以是负整数，也就是说，1px 和 -1px 都是有效的。blur-radius 值必须是正整数。这三个值都可以为 0。

提示　可以使用负的 spread 值，让整个阴影比应用样式的元素小一些（如图 14.5.4 所示）。

提示　颜色可以表示为十六进制数、RGB、RGBA 或 HSLA 值（参见 7.5 节的“CSS 颜色”）。

提示　如果不设置属性值，则使用默认值 none。

提示　inset 关键字可以让阴影位于元素内部。

提示　box-shadow 属性是不继承的。

14.6　应用多重背景

为单个 HTML 元素指定多个背景是 CSS3 引入的一个特性（如图 14.6.1、图 14.6.2 和图 14.6.3 所示）。通过减少对某些元素的需求（这类元素存在只是为了用 CSS 添加额外的图像背景），指定多重背景便可以简化 HTML 代码，并让它容易理解和维护。多重背景几乎可以应用于任何元素。

```
…
<body>
<div class="night-sky">
    <h1>In the night sky…</h1>
</div>
</body>
</html>
```

图 14.6.1　为 class="night-sky" 的单个 div 应用多重背景

```
.night-sky {
    background-color: navy; /* 备用 */
    background-image:
       url(ufo.png), url(stars.png),
       url(stars.png), url(sky.png);

    background-position:
       50% 102%, 100% -150px,
       0 -150px, 50% 100%;

    background-repeat:
       no-repeat, no-repeat,
       no-repeat, repeat-x;

    height: 300px;
    margin: 0 auto;
    padding-top: 36px;
    width: 75%;
}
```

图 14.6.2　首先，为使用旧浏览器的用户定义 background-color。这一步是可选的，但建议添加，从而保证内容的可访问性。然后定义加载和定位背景图像的属性，说明背景图像如何重复

1. 为单个元素应用多重背景图像

(1) 输入 background-color: *b*，这里的 *b* 是希望为元素应用的备用背景颜色（参见图 14.6.4）。

(2) 输入 background-image: *u*，这里的 *u* 是绝对或相对路径图像引用的 URL 列表（用逗号分隔）。

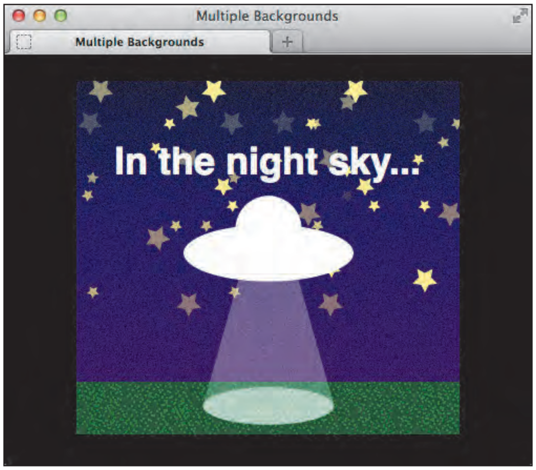


图 14.6.3　支持多重背景的浏览器显示示例的样子。其中，图像是分层次相互重叠在一起的，用逗号分隔的列表中的第一个图像位于顶部。另见彩插

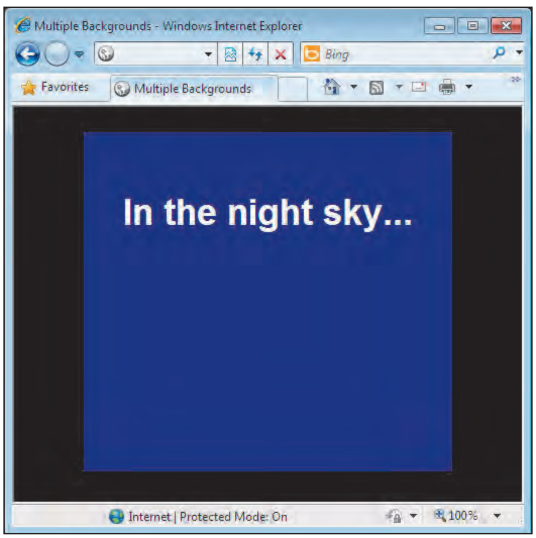


图 14.6.4　IE8 及不支持多重背景图像语法的浏览器会将 background-color 属性指定的背景显示为备用背景颜色

(3) 输入 background-position: *p*，这里的 *p* 是成对的 x-offset 和 y-offset（可以是正的，也可以是负的；带长度单位或者关键字，如 center top）的集合，用逗号分隔。对于第 (2) 步中指定的每个 URL，都应有一组 x-offset 和 y-offset。

(4) 输入 background-repeat: *r*，这里的 *r* 是 repeat-x、repeat-y 或 no-repeat 值，用逗号分隔，第 (2) 步中指定的每个 URL 对应一个值。

14

提示　关于前面各步中列出的属性的更多信息参见 10.10 节。

提示　对于多重背景图像，可以使用标准的短形式语法，即使用逗号分隔每组背景参数，参见图 14.6.5。这种表示方法的好处是开发者既可以指定备用背景颜色，也可以为旧浏览器指定图像，参见图 14.6.6。

提示　指定多重背景不需要使用厂商前缀。

```
.night-sky {
    /* 备用颜色和图像 */
    background: navy url(ufo.png) no-repeat
center bottom;

    background:
        url(ufo.png) no-repeat 50% 102%,
        url(stars.png) no-repeat 100% -150px,
        url(stars.png) no-repeat 0 -150px,
        url(sky.png) repeat-x 50% 100%;

    …
}
```

图 14.6.5　这与图 14.6.2 的代码几乎是相同的，只是使用了简记法，而且为 IE8 及其他旧浏览器同时提供了背景颜色和背景图像（如图 14.6.6 所示）。对于支持这些属性的浏览器，显示效果与图 14.6.6 所示的相同

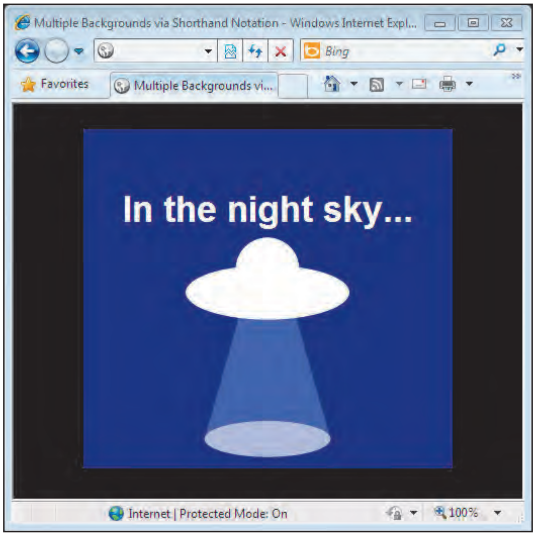


图 14.6.6　旧浏览器现在可以显示比图 14.6.4 更好的效果

14.7　使用渐变背景

渐变背景也是 CSS3 中的新特性，通过它可以在不使用图像的情况下创建从一种颜色到另一种颜色的过渡（参见图 14.7.1 至图 14.7.11）。

```
...
<body>
<div class="vertical-down"><p>default</p>
→ </div>
<div class="vertical-up"><p>to top</p></div>
<div class="horizontal-rt"><p>to right</p>
→ </div>

... 某类div的其余代码 ...
</body>
</html>
```

图 14.7.1　这是所有 CSS 渐变示例（除了图 14.7.10 和图 14.7.11）用到的 HTML。它包含了一系列 div，每个 div 都有一个用于应用渐变样式的类名。（需要说明的是，渐变可以用于大多数元素，并不仅限于 div。）这个页面的完整版本可见本书配套网站

```
.vertical-down { /* 默认 */
    background: silver; /* 备用 */
    background: linear-gradient(silver, black);
}

.vertical-up {
background: silver;
    background: linear-gradient(to top,
    → silver, black);
}
```

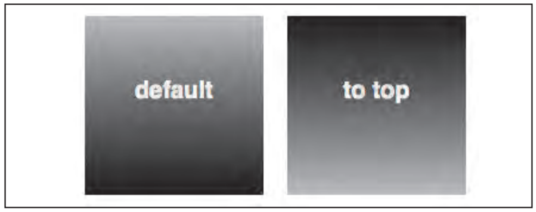


图 14.7.2　CSS 渐变是一个背景图像，因此，用在这里以及其他地方的属性既可以是 background 简记法，也可以是 background-image。应该始终包含一个为旧浏览器准备的基准 background 定义，并放置在渐变背景的定义之前。默认情况下，线性渐变是从上往下渐变的，因此在属性值中不需要指定 to bottom。如果要使用相反的方向，则使用 to top。渐变会依照指定的方向从银灰色渐变为黑色

```
.horizontal-rt {
    background: silver;
    background: linear-gradient(to right,
    → silver, black);
}

.horizontal-lt {
    background: silver;
    background: linear-gradient(to left,
    → silver, black);
}
```



图 14.7.3　指定水平渐变为由左向右渐变还是由右向左渐变

```
.diagonal-bot-rt {
    background: aqua;
    background: linear-gradient(to bottom
    ➝ right, aqua, navy);
}

.diagonal-bot-lt {
    background: aqua;
    background: linear-gradient(to bottom
    ➝ left, aqua, navy);
}

.diagonal-top-rt {
    background: aqua;
    background: linear-gradient(to top
    ➝ right, aqua, navy);
}

.diagonal-top-lt {
    background: aqua;
    background: linear-gradient(to top left,
    ➝ aqua, navy);
}
```

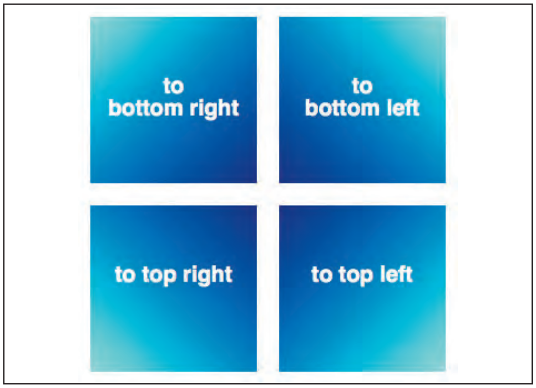


图 14.7.4　有两种指定渐变方向角度的方式。第一种方式就是这里所示的，使用关键字指定渐变方向需要旋转的角度。渐变会从跟指定角相对的角（对角线另一端的角）开始。另见彩插

使用 CSS 创建渐变有两种主要的方式：线性渐变（参见图 14.7.2 至图 14.7.5、图 14.7.10 和图 14.7.11）和径向渐变（参见图 14.7.6 至图 14.7.8），每种方式都有不同的必选参数和可选参数。除非指定一种颜色向另一种颜色过渡的位置，否则浏览器会自行决定不同颜色之间的过渡（参见图 14.7.9）。

```
.angle-120deg {
    background: aqua;
    background: linear-gradient(120deg,
    ➝ aqua, navy);
}

.angle-290deg {
    background: aqua;
    background: linear-gradient(290deg,
    ➝ aqua, navy);
}
```



图 14.7.5　使用关键字（参见图 14.7.4）只能创建沿对角线方向的渐变。第二种方式是指定渐变角度的度数，如 90deg（90 度）。数值代表的是圆周上的点的位置：0 代表最顶端的点，90 代表最左边的点，180 代表最底端的点，270 代表最右边的点。你列出的值决定的是渐变结束的点的位置。因此，0deg 等价于 `to top`，90deg 等价于 `to right`，以此类推。另见彩插

CSS 渐变的语法在相关规范的修订过程中一直在变化。目前的语法是稳定的，但不幸的是仍有几个浏览器使用较旧的语法。因此，为了广泛支持现有的浏览器，还需要使用大量的（甚至可以说是繁琐的）带有厂商前缀的代码（如图 14.7.12 所示）。

了解这一点后，为了降低学习难度，一开始我们将仅演示正确的、无前缀的属性。本节的提示提供了一些额外信息，可供下载的本章代码中包含了完整的示例（包括带厂商前缀的属性）。

根据渐进增强的原则，最好为不支持背景渐变属性的浏览器提供一个备用选项。在 CSS 中，它可以位于背景渐变规则的**前面**，

参见图 14.7.2 ～图 14.7.10。

```
.radial-center { /* 默认 */
    background: red;
    background: radial-gradient(yellow, red);
}
.radial-top {
    background: red;
    background: radial-gradient(at top,
    → yellow, red);
}
```



图 14.7.6　包含额外的可选参数的径向渐变。最简单的例子是默认的样式规则，它使用的参数与线性渐变是一样的。在这个例子中，渐变的原点是元素的中心。可以使用 at top 这样的关键字指定中心的位置。跟往常一样，可以在径向渐变声明之前为较旧的浏览器提供一个背景的定义。另见彩插

```
.radial-size-1 {
    background: red;
    background: radial-gradient(100px 50px,
    → yellow, red);
}
.radial-size-2 {
    background: red;
    background: radial-gradient(70% 90% at
    → bottom left, yellow, red);
}
```

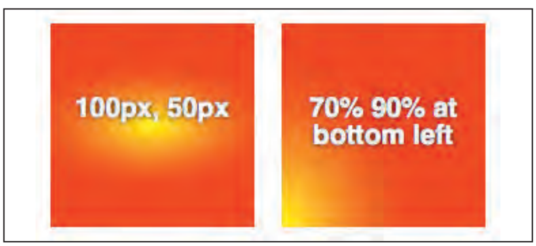


图 14.7.7　还可以控制渐变的尺寸。这里展示了几个例子。另见彩插

```
.radial-various-1 {
    background: red;
    background: radial-gradient(closest-side
    → at 70px 60px, yellow, lime, red);
}
.radial-various-2 {
    background: red;
    background: radial-gradient(30px 30px
    → at 65% 70%, yellow, lime, red);
}
```

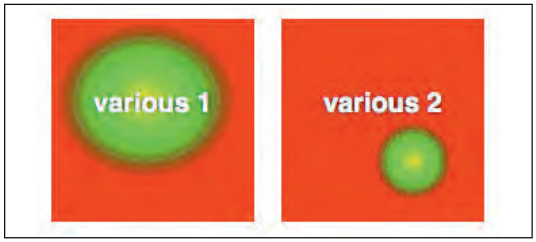


图 14.7.8　这些例子结合了几个参数。它们都有三个颜色。位于 at 后面的值指定的是渐变的中心坐标。在第二个例子中，30px 30px 标记了渐变的尺寸。在第一个例子中，closest-side 决定了渐变的尺寸，它告诉浏览器从中心（指定为 at 70px 60px）向包含该渐变的区域的最近的一边伸展。另见彩插

```
.color-stops-1 {
    background: green;
    background: linear-gradient(yellow 10%,
    → green);
}
.color-stops-2 {
    background: green;
    background: linear-gradient(to top right,
    → yellow, green 70%, blue);
}
```

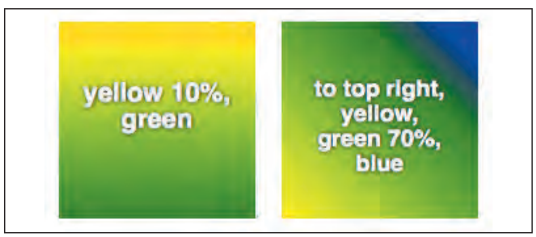


图 14.7.9　可以使用百分数指定一个以上的颜色停止位置。另见彩插

1. 创建备用背景颜色

输入background: *color*或者background-color: *color*，这里的*color*可以是十六进制数、RGB值以及其他任何支持的颜色名称，另外也可以使用图像。最好不要将RGBA、HSL或HSLA值作为备用背景颜色，因为IE8及以前的版本不支持。

2. 定义线性渐变

(1) 输入background: linear-gradient(。

(2) 如果你希望渐变的方向是从上往下（默认方向），则跳过这一步。

输入**方向**后面加一个逗号，这里的方向指的是to top、to right、to left、to bottom right、to bottom left、to top right或to top left这样的值。

或者输入**方向**后面加一个逗号，这里的方向指的是角度值（如45deg、107deg、180deg或310deg）。

(3) 根据后面讲到的“指定颜色”或“指定颜色和颜色的停止位置”，定义渐变的颜色。

(4) 输入);完成渐变。

3. 定义径向渐变

(1) 输入background: radial-gradient(。

(2) 指定渐变的形状。如果你希望渐变的形状根据第(3)步中指定的尺寸自行确定，则跳过这一步。否则，输入circle或ellipse。（注意这一声明在某些情况下会被忽略。）

(3) 指定渐变的尺寸。如果你希望渐变的尺寸为自动指定的值（默认值为farthest-corner，最远的角），则跳过这一步。

输入*size*，这里的*size*可以是同时代表渐变宽度和高度的一个长度值（如200px或7em），也可以是代表宽度和高度的一对值（如390px 175px或60% 85%）。注意，如果只使用一个值，则这个值不能是百分数。

或者输入*size*，这里的*size*是closest-side、farthest-side、closest-corner或farthest-corner。这些关键字代表相对于渐变的中心，渐变可以伸展到多大的空间。边界决定了渐变的尺寸。

(4) 指定渐变的位置。如果你希望渐变从元素的中心开始（默认值），则跳过这一步。

输入*pos*，这里的*pos*是at top、at right、at left、at bottom right、at bottom left、at top right或at top left等表示渐变中心位置的值。

或者输入*pos*，这里的*pos*是表示渐变中心位置的一对坐标，并以at开头。例如at 200px 43px、at 33% 70%、at 50% -10px等。（仅指定一对。）

(5) 如果指定了第(2)步至第(4)步中的任何一步，输入一个逗号。需要说明的是，如果指定了这些步骤中的多个值，这些值之间不需要以逗号分隔。

(6) 根据后面讲到的“指定颜色”或“指定颜色和颜色的停止位置”，定义渐变的颜色。

(7) 输入);完成渐变。

4. 指定颜色

输入至少两种颜色，每个颜色之间用逗号分隔。指定的第一个颜色出现在渐变的开始位置，最后一个颜色出现在渐变的结束位置。对于径向渐变，它们分别为最里边的颜色和最外边的颜色。

颜色值可以是十六进制数、RGB、RGBA、HSL或HSLA值的组合。除非指定一种颜色向另一种颜色过渡的位置，否则浏览器会自行决定不同颜色之间的过渡。

5. 指定颜色和颜色的停止位置

根据“指定颜色”中的说明进行操作。如果需要，对每个颜色值指定一个百分

数以控制颜色出现在渐变中的位置（参见图 14.7.9）。这个值可以是负数，也可以是大于 100% 的数。

```
.about {
    /* 备用 */
    background-color: #ededed;

    background-image:
    → linear-gradient(#fff, #ededed);
    border: 1px solid #d0d0d0;
    -webkit-box-shadow: 0 0 3px #d0d0d0;
    box-shadow: 0 0 3px #d0d0d0;
    padding: 0.313em 0.625em 0.625em;
}
```

图 14.7.10　我们为曾经用过的 About Me 模块添加了新样式，会是什么样子呢？可以注意到，这里用 background-image 代替了 background 简记法（结果是完全一样的）



图 14.7.11　现在它在先前添加的阴影和边框样式的基础上，又有了一个简单的线性渐变

```
.vertical-down {
    background: silver;

    /* Chrome 10-25, iOS 5+, Safari 5.1+ */
    background: -webkit-linear-gradient
    → (top, silver, black);

    /* Firefox 3.6-15 */
    background: -moz-linear-gradient
    → (top, silver, black);

    /* Opera 11.10-12.00 */
    background: -o-linear-gradient
    → (top, silver, black);

    /* 标准语法，支持浏览器：
       Chrome 26+, Firefox 16+, IE 10+,
       → Opera 12.10+
    */
    background: linear-gradient(silver,
    → black);
}
```

图 14.7.12　噢！这是让图 14.7.2 中的渐变样式规则在各种支持该样式的浏览器里都能正常工作所需要的代码。这里没有包含针对 Safari 4 ～ 5 和 Chrome 10 之前版本的声明，如果加上它们，代码就更长了。大部分这样的代码都可以通过 CSS3, Please!（www.css3please.com）生成（同时生成关于浏览器支持情况的注释）

6. 为旧版浏览器创建渐变代码

先前我曾指出，要让渐变可以在各种支持它的浏览器上都能正常显示，哪怕是最简单的渐变，都需要一大堆代码。现在你可以看看这些代码（如图 14.7.12 所示）。没有人愿意手写这么多代码，哪怕知道怎么写。幸好有很多工具可以帮我们做这个工作（参见提示）。

好的方面是，大多数需要厂商前缀的浏览器过一段时间（不用太久）就会因为太旧而无需再为其添加额外的代码。

提示 简单起见，在大多数示例中我都使用了颜色关键字。在实践中，你可以根据自己的偏好使用十六进制数或其他的颜色值。

提示 关于对最新浏览器的支持情况，参见 http://caniuse.com/#search=gradient。

提示 渐变具有很强的通用性。可以为一个背景定义多个渐变，每个定义之间使用逗号分隔。通过定义多个渐变，可以创造出很多有趣的效果。不妨访问 Lea Verou 的 CSS3 Patterns Gallery（http://lea.verou.me/css3patterns/）看看有哪些有趣的效果。

提示 可以使用 Microsoft 提供的 CSS 渐变背景制作器（http://ie.microsoft.com/testdrive/graphics/cssgradientbackgroundmaker/）这样的可视化工具替代繁琐的代码编写工作。这个工具还可以帮你创建全部的厂商前缀属性，让渐变获得最大程度的浏览器支持。ColorZilla 的渐变生成器（http://colorzilla.com/gradient-editor/）也是类似的，但需要注意的是通常它生成的代码要比实际需要的多得多。

提示 可以通过指定 background-color 或 background-image 为不支持渐变的浏览器提供后备方案，不过要记住的是，无论浏览器是否使用 CSS 中的图像，这些图像都会被下载。

14.8 为元素设置不透明度

使用 opacity 属性可以修改元素（包括图像）的透明度，如图 14.8.1 ~ 图 14.8.5 所示。

修改元素不透明度的方法

输入 opacity: *o*，这里的 *o* 表示元素的不透明程度（两位小数，不带单位）。

```
...
<body>

<div class="box">
    <img src="img/sleeves.jpg" width="420"
    → height="296" alt="Record sleeves" />
</div>

</body>
</html>
```

图 14.8.1 这个文档包含一个 div，其中有一个图像

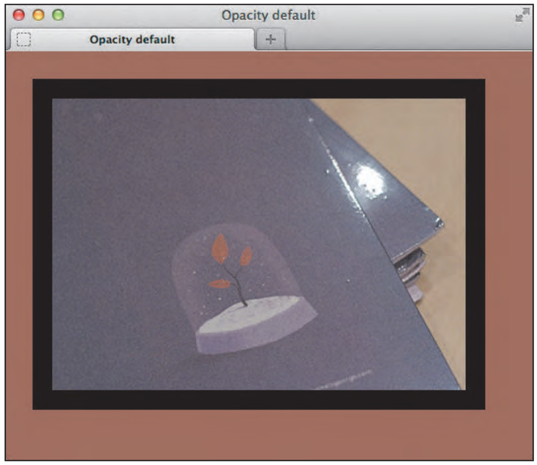


图 14.8.2 这是图 14.8.1 中 div 的默认样式。同其他元素一样，浏览器为其设置的默认样式为 opacity: 1;（不需要在样式表中指定）。该 div 有一个黑色的背景，由于该元素有内边距，因此背景会显示出来。body 的背景为褐色。另见彩插

```
.body {
    background: #a2735f; /* brown */
}

.box {
    background: #000; /* black */
    opacity: .5;
    padding: 20px;
    width: 420px;
}
```

图 14.8.3 将 opacity 设为小于 1 的值可以让元素及其子元素变成半透明的样子。在这个例子中，不透明度被设为 50% 或 0.5，小数点前的 0 是可选的

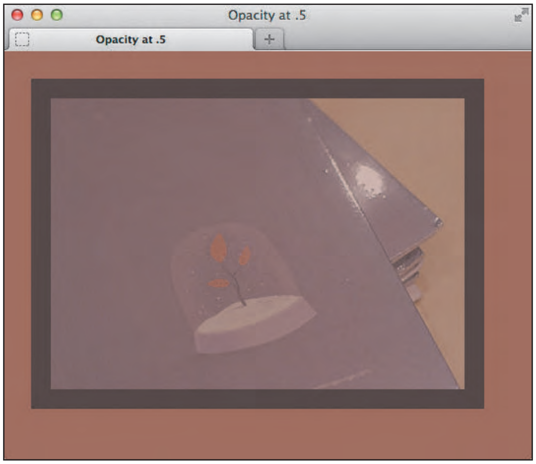


图 14.8.4　这是示例中 div 元素的 opacity 设为 0.5（即不透明度为 50%）时的样子。注意 div 元素的黑色背景现在呈现为深褐色，图像也变成半透明的，且带有一些褐色的色彩。如果将某元素的 opacity 设为小于 1 的值，包含它的周边元素就会透出来。在这个例子中，透出来的是 body 的褐色背景。如果将 body 的背景设为红色，效果会更明显（参见图 14.8.5）。另见彩插

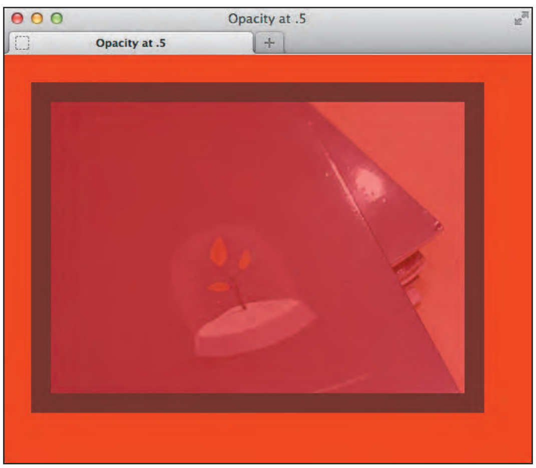


图 14.8.5　示例中 div 仍设为 opacity: .5;，但 body 改为 background-color: red;。通透的程度取决于不透明度。如果将不透明度设为 0.35，就会透出更多的红色；如果设为 0.8，则透出的红色就更少。另见彩插

```
div {
    -ms-filter: progid:DXImageTransform.
    ➝ Microsoft.Alpha(opacity=50);
    filter: alpha(opacity=50);
    opacity: .5;
    zoom: 1;
}
```

图 14.8.6　这是针对旧版 IE 应用 IE 专有滤镜设置不透明度的代码。zoom: 1; 也是 IE 专有的，该语句也是用来实现效果的一部分。现代浏览器仍会使用 opacity: .5; 的设置。注意 IE 滤镜使用的是不同的数字。在这个例子中，使用 50 代替 0.5，表示 50% 的不透明度

提 示　opacity 的默认值为 1，参见图 14.8.2。该属性值可以使用 0.00（完全透明）至 1.00（完全不透明）之间的两位小数。例如，opacity:.09;、opacity: .2; 或者 opacity: .75;。小数点前后的 0 可有可无，如 0.00 可以写作 0，1.00 可以写作 1。

提 示　文本也会受不透明度的影响。如果 div 中有白色文本（只是打个比方），图 14.8.4 中的文字会带有一定程度的棕色，图 14.8.5 中的文字会带有一定程度的红色。

提 示　通过使用 opacity 属性和 :hover 伪属性，可以产生一些有趣且实用的效果。例如，img { opacity: .75; } 默认情况下可以将图像设置为 75% 的不透明度，img:hover { opacity: 1; } 可导致用户鼠标停留在元素上时元素变为不透明。在将缩略图链接到全尺寸版本时经常看到这种效果。对于访问者来说，悬浮可增强图像的动感。

提 示　opacity 属性与使用 RGBA 或 HSLA 设置的 alpha 透明背景色是两个容易混淆的概念。如图 14.8.4 和图 14.8.5 所示，opacity 影响的是整个元素（包括其内容），而 background-color: rgba(128,0,64,.6); 这样的设置仅影响背景的透明度。参见 7.5 节“CSS 颜色”中关于 RGBA 的示例。

提 示　Internet Explorer 9 之前的版本并不原生支持 opacity 属性，但对于它们，可以使用专有的滤镜效应，参见图 14.9.6。确保不要在页面中过多使用这类滤镜，因为它们会反过来影响浏览器的性能。

提 示　无论显示成什么样子，opacity 属性并不是继承的。opacity 的值小于 1 的元素的子元素也会受到影响，但这些子元素的 opacity 值仍为 1。

14.9　生成内容的效果

使用 :before 和 :after 伪元素可以很方便地为页面添加一些令人难以置信的设计效果。它们可以与 content 属性结合使用，从而创建所谓的**生成内容**（generated content）。生成内容指的是通过 CSS 创建的内容，而不是由 HTML 生成的。

不要误解，生成内容并不是为页面添加段落或标题。这些内容还是应该交给 HTML。使用生成内容可以添加符号（如图 14.9.1 至图 14.9.4 所示），创建用于添加样式的空内容元素（如图 14.9.5 至图 14.9.8 所示），还可以做很多其他的事情（参见提示）。

我本可以在第一个例子的 HTML（参见图 14.9.1）中包含箭头，但这些箭头属于样式，以后有可能发生改变。如果需要更改样式，只需要修改 .more 类（参见图 14.9.2）即可，而不需要改动可能数以百计的 HTML 页面。

```
…

<pThis area is one of the most tranquil
→ spaces at the Villa. As I wandered around,
→ enjoying shade provided by sycamore and
→ laurel trees and serenaded by splashing
→ water from two sculptural fountains,
→ I couldn't help but think &hellip;
→ <a href="victoria.html" class="more">
→ Read More</a></p>

…
```

图 14.9.1　一个简单的段落，其中有一个带类的链接。注意 Read More 的后面没有两个箭头

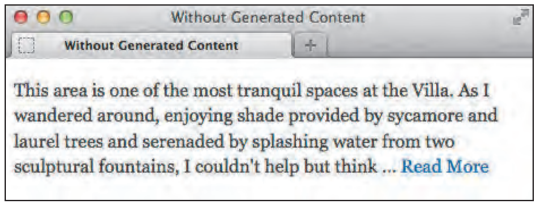


图 14.9.2　使用基本 CSS 文本样式的段落。链接后面没有箭头

```
…

.more:after {
    content: " »";
}
```

图 14.9.3　发挥魔力的地方

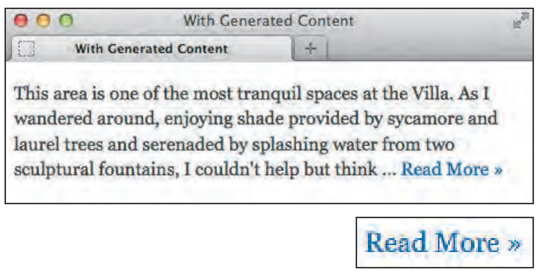


图 14.9.4　现在，带有 class="more" 的元素会在其后显示一个双箭头

```
...
<div class="travels">
    <h2>My Travels</h2>
    <img src="img/map.jpg" class="map"
    ➝ tabindex="0" ... />

    <ul class="cities clearfix">
       <li>Victoria</li>
       <li>Los Angeles</li>
       <li>Mexico City</li>
       <li>Buenos Aires</li>
       <li>Paris</li>
       <li>Kampala</li>
       <li>Lagos</li>
       <li>Cairo</li>
       <li>Beijing</li>
    </ul>
</div>
...
```

图 14.9.5　这个城市列表会显示为底部带有箭头标记的方形气泡

第二个例子使用 `content: " ";` 创建了一个空格，并使用 CSS 边框创建了一个三角形样式，如图 14.9.6 所示（参见提示）。可以对生成内容添加背景颜色、背景图像（包括渐变）、宽度、高度、圆角等样式。

提示　完整的示例代码位于 www.htmlcssvqs.com/8ed/14/gencon。

提示　可以对单个元素同时应用 `:before` 和 `:after`，从而建立两个可供施加样式的额外容器。例如，可以应用一种不同于 14.6 节所介绍的多重背景效果。Chris Coyier 演示了使用 `:before` 和 `:after` 的更多神奇用法（http://css-tricks.com/pseudo-element-roundup/）。

提示　Joel Glover 深入讨论了如何使用边框样式创建三角形，参见 http://appendto.com/blog/2013/03/pure-css-triangles-explained/。

```
.travels {
    /* 让位于其中的.cities可以使用绝对定位 */
    position: relative;
}

.map:focus + .cities,
.map:hover + .cities {
    left: 50%; /* 显示 */
}

/* 气泡 */
.cities {
    background: #2B2B2B;
    border-radius: 5px 5px;
    left: -999em; /* 隐藏 */
    margin-left: -111px;
    padding: .5em 0 .9375em .9375em;
    position: absolute;
    top: -75px;
    width: 222px;
}

/* 气泡下的三角形 */
.cities:after {
    border: solid transparent;
    border-top-color: #2b2b2b;
    border-width: 15px;
    content: " "; /* 空格 */
    height: 0;
    left: 50%;
    margin-left: -15px;
    position: absolute;
    top: 99.9%;
    width: 0;
}

...
```

图 14.9.6　这些代码会产生一些效果。最后一条规则使用 `:after` 伪元素创建一个三角形并定位于城市列表的下方。第三条规则让列表在默认情况下位于屏幕之外。第二条规则让列表在访问者与地图交互的时候显示出来（如图 14.9.7 和图 14.9.8 所示）。它是绝对定位的，不过是相对于在 HTML 中包含它的 `.travels div` 的（参见图 14.9.5）

提示　Simon Højberg 的 CSS Arrow Please!（http://cssarrowplease.com/）是一个帮你创建 CSS 三角形代码的工具。

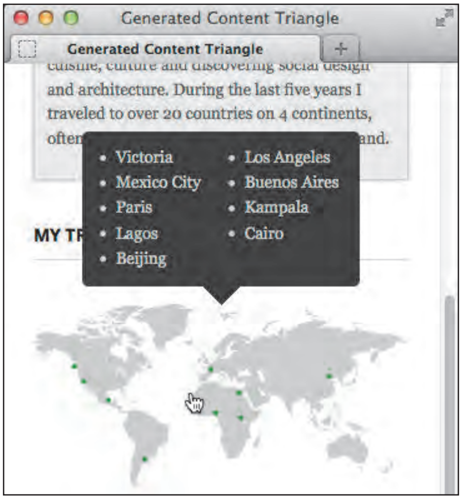


图 14.9.7　气泡底部的箭头是纯 CSS，不是图像！当鼠标移到地图的任何位置，城市气泡就会显示出来。不过，这还不是全部……

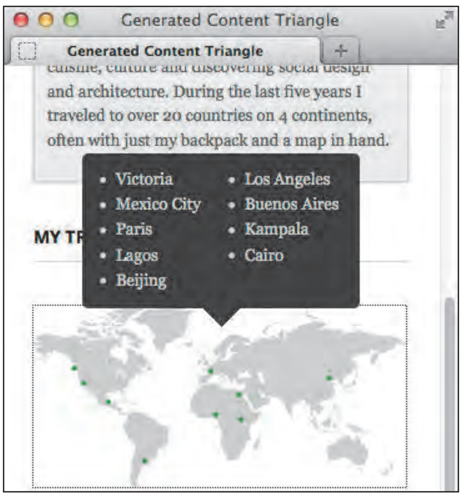


图 14.9.8　img 里的 tabindex="0" 属性（参见图 14.9.5）让 :focus 伪选择器（参见图 14.9.6）得以正常工作，如果访问者使用键盘的 Tab 键（而不是通过鼠标）将焦点放到地图上，也可以让城市气泡显示出来。这也体现了可访问性的实现（只有使用 Tab 键将焦点放到地图上，才会显示图中所示的虚线）

提 示　:focus 伪类默认对链接和表单元素起作用。对于这些元素，不必添加 tabindex="0"。

提 示　第 11 章演示的 .clearfix 类就使用了生成内容。

14.10　使用 sprite 拼合图像

浏览器通常很快就可以将文本显示出来，但图像往往会减慢页面的加载速度。这一现象在移动设备上尤为明显。为了解决这一问题，可以将多个图像拼合成单个背景图像（sprite），再通过 CSS 控制具体显示图像的哪一部分。其中的秘密就是 background-position 属性。

图 14.10.1 显示了一个标准的无序列表（参见第 15 章）。我们的目标是在每个文档链接前面显示一个来自图 14.10.2 所示的 sprite 的图标。

```
…
<ul class="documents">
    <li><a href="expenses.xls" class="icon">
    ➝ Business expenses</a></li>
    <li><a href="user-manual.pdf"
    ➝ class="icon">User Manual</a></li>
    <li><a href="story.docx" class="icon">
    ➝ Short story</a></li>
    <li><a href="brochure.pdf" class="icon">
    ➝ Vacation brochure</a></li>
</ul>
…
```

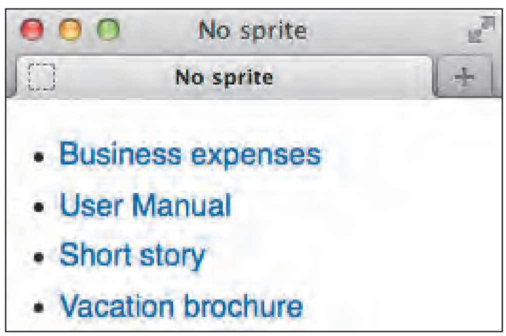


图 14.10.1　可以根据文件扩展名和 .icon 类找到对应该文档类型的图标。在做这些工作之前，列表前面显示默认的圆点

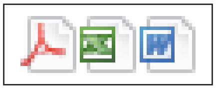


图 14.10.2　这个 sprite 图像包含三个图标，每个都是 16 像素 ×16 像素大小，相邻两个之间有 2 像素的空隙

可以将 sprite 应用于任意数量的元素。在这个例子中，我使用 .icon:before 来实现所需的效果。这样，sprite 就是通过 content: " "; 生成的空格的背景图像。将其设置为 display: block;，从而可以设置与图标大小匹配的高度和宽度。没有这三个属性，图像将不会显示。通过使用 background-position（如图 14.10.3 所示），可以将正确的图标放入该位置（如图 14.10.4 所示）。

```
...
.documents { list-style: none; }

.icon {
    display: inline-block;
    min-height: 16px;
    padding-left: 23px;
    position: relative;
    ...
}

.icon:before {
    background-image: url(sprite.png);
    content: " ";
    display: block;
    height: 16px; /* 图标高度 */
    position: absolute;
    width: 16px; /* 图标宽度 */
}

a[href$=".xls"]:before {
    background-position: -17px 0;
}

a[href$=".docx"]:before {
    background-position: -34px 0;
}
```

图 14.10.3　也可以对 HTML 中所有相关的 a 元素使用 class="pdf"、class="xls" 等（参见图 14.10.1）。不过，根据 href 值中的链接就能确定文档的类型。这样，通过使用 $= 就可以根据特定的扩展名找到对应的类型



图 14.10.4　每个链接前面都显示出了正确的图标

提 示　每个 sprite 都可以包含尺寸不同的图像，这些图像之间的距离也可以不同。可以将它们水平排列，也可以竖直排列。

提 示　使用 sprite 并不一定要使用 :before 或 :after。可以将 sprite 背景直接用于元素本身。

提 示　可以在链接或其他元素的 :hover 状态中修改 sprite 的 background-position 以实现翻转效果。

提 示　Project Fondue 提供的 CSS Sprite Generator（http://spritegen.website-performance.org）是一个帮助创建 sprite 和背景定位的工具（有很多类似的工具）。

提 示　本节使用的图标来自 Mark James 提供的免费 Silk 图标（www.famfamfam.com/lab/icons/silk/）。

提 示　可以创建为 Retina 显示屏或其他高像素密度显示屏准备的 sprite，参见 12.5 节。

第 15 章

列表

15

本章内容

- ❑ 创建有序列表和无序列表
- ❑ 选择标记
- ❑ 使用定制的标记
- ❑ 选择列表的起始编号
- ❑ 控制标记的位置
- ❑ 同时设置所有的列表样式属性
- ❑ 设置嵌套列表的样式
- ❑ 创建描述列表

HTML 包含专门用于创建项目列表的元素。你可以创建普通列表、编号列表、符号列表以及描述列表，可以在一个列表中嵌套另外一个或多个列表。

所有的列表都是由父元素和子元素构成的。父元素用于指定要创建的列表的类型，子元素用于指定要创建的列表项目类型。下面列出了三种列表类型以及组成它们的元素。

- ❑ 有序列表，ol 为父元素，li 为列表项。
- ❑ 无序列表，ul 为父元素，li 为列表项。
- ❑ 描述列表，dl 为父元素，dt 和 dd 分别代表 dl 中的术语和描述。描述列表在 HTML5 之前称为**定义列表**。

在这些类型里面，无序列表是万维网上最为常见的列表类型，它也是对大多数类型的导航进行标记的事实标准（本书有几个这样的例子）。不过，上述三种类型都各有用武之地，本章将对此进行讲解。

15.1 创建有序列表和无序列表

如果列表项的顺序对于列表来说非常关键，那么这种情况有序列表就是恰当的选择。有序列表适于提供完成某一任务的分步说明（如图 15.1.1 和图 15.1.2 所示），或用于创建大型文档的大纲。总之，它适用于任何强调顺序的项目列表。

```
...
<body>

<h1>Changing a light bulb</h1>

<ol>
    <li>Make sure you have unplugged the
    ➝ lamp from the wall socket.</li>
    <li>Unscrew the old bulb.</li>
    <li>Get the new bulb out of the
    ➝ package.</li>
    <li>Check the wattage to make sure
    ➝ it's correct.</li>
    <li>Screw in the new bulb.</li>
    <li>Plug in the lamp and turn it
    ➝ on!</li>
</ol>

</body>
</html>
```

图 15.1.1 目前还没有对列表标题进行格式化的正式方法。大多数情况下，使用常规的标题或段落即可，就像下面的例子那样。按照惯例（并非必须），可以对列表项目进行缩进，表明它们是嵌套在 ol 里面的（对于 ul 也是这样的）。不过，这些缩进在页面中不会显示出来，页面中的显示是由应用到列表上的 CSS 控制的

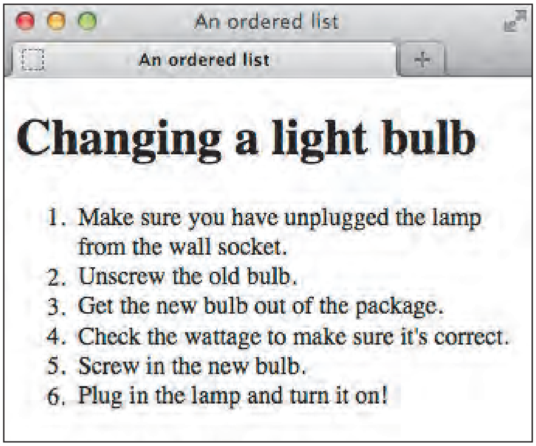


图 15.1.2　这个列表使用默认的阿拉伯数字创建带编号的有序列表。可以使用 CSS 对此进行修改。有序列表和无序列表在显示时默认都会进行缩进，无论它们在 HTML 中是否有缩进（参见图 15.1.1）

无序列表恰好相反，且应用更为普遍，如果列表项的顺序不太重要就要使用无序列表，参见图 15.1.3 ～图 15.1.7。

两种列表类型都适于标记特定类型的导航，参见第二条提示。

```
...
<body>

<h1>Product Features</h1>
<ul>
    <li>One-click page layout.</li>
    <li>Spell-checker for 327
    languages.
    → </li>
    <li>Image retouching tool.</li>
    <li>Unlimited undos and redos.</li>
</ul>

</body>
</html>
```

图 15.1.3　无序列表中的项目与有序列表中的是相同的，只有 ul 元素是不同的

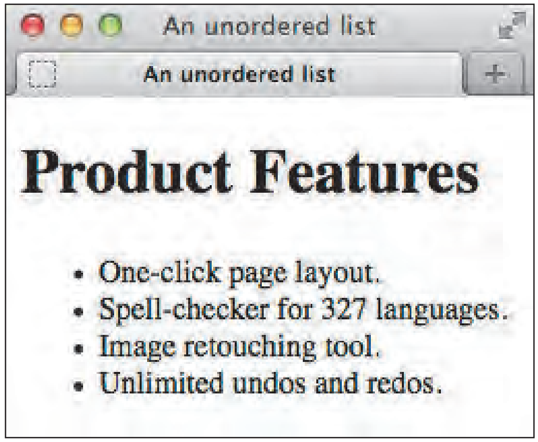


图 15.1.4　在默认情况下，无序列表前面显示实心的圆点。可以通过 CSS 对此进行修改

```
...
<h1 class="hdg">Product Features</h1>
<ul>
    <li>One-click page layout.</li>
    ...
</ul>
...
```

图 15.1.5　正如在图 15.1.4 中所看到的，元素之间的默认间距并不理想。接下来将演示如何对该样式进行调整。这里的 HTML 与图 15.1.3 中的是相同的，只是为 h1 添加了一个类（这也是使用类方便的原因）。添加了图 15.1.6 中的样式之后，就可以为任何刚好位于 ul 或 ol 前面的标题添加 .hdg 类，并得到相同的效果。类似地，可以为列表创建类，从而让缩进和间距的样式规则可以用于 ol 或 ul

创建列表

(1) 输入 （有序列表）或 （无序列表）。对于有序列表，可以包含 start、type 和 reversed 这三个可选的属性。（关于 start，参见 15.4 节；关于 type，参见 15.2 节；关于 reversed，参见最后一条提示。）

(2) 输入 <li> 以开始第一个列表项目。对于有序列表，可以包含可选的 value 属性（更多细节参见 15.4 节）。

(3) 添加要包含在列表项目内的内容（如文本、链接、img 元素等）。

(4) 输入 以结束列表项目。

(5) 对于每个新的列表项目，重复第 (2) 步至第 (4) 步。

(6) 输入 或 （与第 (1) 步中的开始标签对应）以结束列表。

```
body {
    font-family: sans-serif;
}

.hdg {
    font-size: 1.5em;
    margin-bottom: 0;
}

.hdg + ul,
.hdg + ol { /* 用于ul或ol */
    margin-top: .5em; /* 约8px */
}

ul { /* 减少缩进 */
    margin-left: 0; /* 针对IE7及以下版本 */
    padding-left: 18px; /* 约1.125em */
}

ul li {
    /* 列表项之间的空格 */
    margin-top: .4em; /* 约6px */
}
```

图 15.1.6　第二条和第三条规则定义了标题与列表之间的间距，最后一条规则定义了每个列表项目之间的间距。第四条规则减小了缩进，从而让列表标记更靠近左边缘。padding-left: 18px; 为列表标记腾出了位置。如果该属性值为 0，则文本会对齐左边缘，而列表标记就看不见了，因为标记在默认情况下是位于列表项目之外的。不一定非要使用 em 为单位，如果用 em 的话，要记住 em 值是相对于父元素的字体大小的

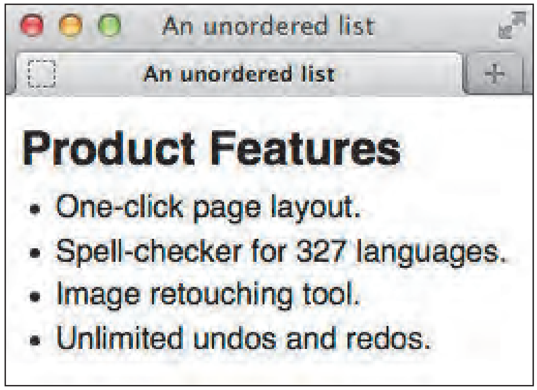


图 15.1.7　页面的呈现效果好了很多

提示　不要根据希望添加在内容旁边的标记样式决定要使用的列表类型，毕竟，这些标记的样式随时都可以通过 CSS 进行修改（甚至可以在有序列表上显示符号）。相反，应该考虑列表的含义——列表会随着其中项目顺序的改变而改变吗？如果答案是肯定的，就使用有序列表进行标记。否则，就使用无序列表。

提示　使用列表标记链接组时，大多数情况下均可以使用无序列表，如主导航链接、指向一组视频或相关报道的链接、页脚中的一组链接等。对于面包屑导航则应使用有序列表，因为这些链接有特定的次序（换句话说，顺序是有意义的）。面包屑导航通常水平地显示在主要内容区域的上方，指示当前页面在网站导航路径中的位置。分页标记是链接的水平列表，看起来像 1 | 2 | 3 | 4，通常会出现搜索结果和产品的列表，允许你在不同的结果页面之间跳转。图 15.7.5 包含了主导航和面包屑的例子。

提示　第 11 章和第 12 章的完整示例网页演示了使用和呈现列表的各种方式。其中有用于导航和其他链接组的无序列表、用于一系列链接到之前的博客条目的有序列表。第 3 章也有在导航中使用 ul 的例子。

提示　除非使用 CSS 另行指定，有序列表中的项目使用阿拉伯数字（1、2、3 等）进行编号（参见图 15.1.2）。

提示　默认情况下，无序列表的项目前面显示实心的圆点（参见图 15.1.4）。你可以选择不同的符号（参见 15.2 节），甚至可以自定义符号（参见 15.3 节）。

提示　列表在默认情况下是从左边缩进的，不过可以通过 CSS（参见图 15.1.6）取消缩进、减小缩进（参见图 15.1.7）或增加缩进。15.3 节也演示了这些操作。列表标记可能会显示在内容的外面，也可能因为超出窗口的边缘而消失（参见图 15.1.6 所示的内边距为零的样式）。

提示　默认情况下，列表有一定的左侧外边距，从而形成缩进。可以使用 CSS 取消（减小）或增大缩进，参见图 15.1.6 和图 15.1.7。15.3 节也会解释这一点。根据减少缩进的量，符号可能处于内容的外边或者消失在窗口的左边缘之外（参见图 15.1.6 中假设内边距为 0 的情况）。

提示　可以在一个列表中创建另一个列表（即**嵌套列表**，nesting list），甚至可以混合或者搭配使用有序列表和无序列表。不过，要确保用到所有需要的开始标签和结束标签，对每个列表进行正确地嵌套。嵌套有序列表和无序列表的例子见 15.7 节。

提示　只能将列表内容放在 li 元素里。例如，不允许将内容放置在 ol 或 ul 开始标签和第一个 li 元素之间。li 元素里面可以放置多种类型的元素，如各种短语内容元素（em、a、cite 等）。在列表项目中嵌套段落、div 这样的元素也是有效的。

提示　如果将内容方向指定为从右向左（例如在语言为希伯来语的情况下），列表就会通过右侧外边距进行缩进。要指定内容方向，可以在页面的 html 元素中设置 dir 属性，如 <html dir="rtl" lang="he">。在这个例子中，lang 被设为表示希伯来语的 he。还可以在 body 里面的元素上设置 dir 和 lang 以覆盖 html 元素上的设置。dir 属性的默认值为 ltr。

提示　截至本书写作之际，浏览器对布尔型的 reversed 属性的支持还不够理想：只有 Chrome 18+、Firefox 18+ 和 Safari 5.2+。该属性的作用是指示降序排列的有序列表（使用 <ol reversed> 或 <ol reversed="reversed"> 均可指定该属性）。支持该属性的浏览器会自动颠倒列表的编号。

15.2　选择标记

创建列表时，无论是有序列表（如图 15.2.1 所示）还是无序列表，都可以选择出现在列表项目左侧的标记的类型。

1. 选择标记

在样式表中，输入 list-style-type: *marker*，这里的 *marker* 是以下属性值中的一种。

- disc（圆点，●）
- circle（圆圈，○）
- square（方块，■）

- decimal（数字，1、2、3……）
- upper-alpha（大写字母，A、B、C……）
- lower-alpha（小写字母，a、b、c……）
- upper-roman（大写罗马数字，Ⅰ、Ⅱ、Ⅲ、Ⅳ……），如图 15.2.2 和图 15.2.3 所示
- lower-roman（小写罗马数字，i、ii、iii、iv……）

```
…
<body>

<h1 class="hdg">The Great American Novel</h1>
<ol>
    <li>Introduction</li>
    <li>Development</li>
    <li>Climax</li>
    <li>Denouement</li>
    <li>Epilogue</li>
</ol>

</body>
</html>
```

图 15.2.1 这是用做示例的有序列表。我们将对其应用大写罗马数字（upper-roman）。注意，为了利用图 15.1.6 中的 .hdg 规则，我对 h1 应用了 class="hdg"

```
li {
    list-style-type: upper-roman;
}

/* 下面的代码的效果一样，因为li元素继承list-
   style-type属性

ol {
    list-style-type: upper-roman;
}
*/
```

图 15.2.2 可以对 ol（参见注释）、ul 或者列表项本身应用 list-style-type 属性，如下面的代码所示。虽然这里没有显示，但我为 body 元素应用了某种字体

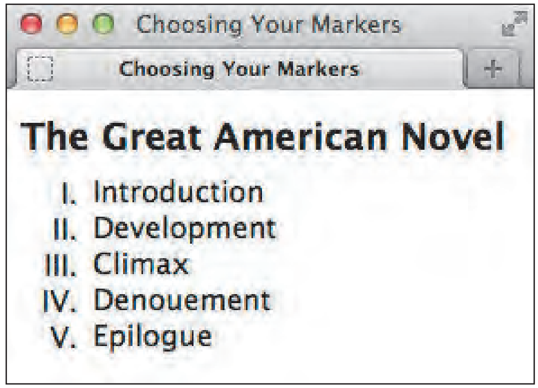


图 15.2.3 现在，有序列表采用大写罗马数字进行编号。注意，大多数浏览器会让编号右对齐

2. 显示无标记列表

在样式表规则中，输入 list-style-type: none。

提 示 可以通过 list-style-type 对 ol 和 ul 设置任何标记样式。换句话说，可以对 ol 使用方块标记，也可以对 ul 使用数字标记。

提 示 还有大量的其他标记类型可用，不过浏览器对不同标记类型的支持情况不同。

提 示 也可以在 HTML 中通过 type 属性为有序列表指定标记类型，但推荐的做法是尽可能在 CSS 中定义列表样式类型。一种例外情况请参见 html5doctor.com/ol-element-attributes/。type 属性可接受的值包括 A、a、I、i 和 1（默认值为 1）。例如，<ol type="I"> 表示使用大写罗马数字作为编号。

15.3 使用定制的标记

如果对使用圆圈、方块、圆点以及罗马数字这些标记感到厌倦，也可以使用图像创建自己定制的标记。要使用定制的标记，不

必修改 HTML（如图 15.3.1 所示），仅需修改 CSS（参见图 15.3.2 ~ 图 15.3.7）。

```
…
<body>

<h1 class="hdg">Product Features</h1>
<ul>
    <li>One-click page layout.</li>
    <li>Spell-checker for 327 major
    → languages.</li>
    <li>Image retouching tool.</li>
    <li>Unlimited undos and redos.</li>
</ul>

</body>
</html>
```

图 15.3.1　这个列表与任何普通无序列表都是相似的，不过，加上一些 CSS 后，它看起来就不一样了

使用定制的标记

(1) 在目标列表或列表项的样式规则中，输入 `list-style: none;` 以取消常规的标记。

(2) 在目标列表的样式规则中，设置 `margin-left` 和 / 或 `padding-left` 属性，指定列表项目缩进的大小。为了在不同的浏览器上实现相似的效果，通常需要同时设置这两个属性（参见图 15.3.2 和图 15.3.6）。注意，如果为内容设置了 `dir="rtl"`，那么就应该设置 `margin-right` 和 `padding-right` 属性。关于在列表中使用 `dir`、`lang` 及从右至左的语言，参见 15.1 节的提示。

(3) 在目标列表的 `li` 元素的样式规则中，输入 `background: url(`*`image.ext`*`)` *`repeat-type horizontal vertical`*`;`，其中 *`image.ext`* 是要作为定制标记的图像的路径和文件名，*`repeat-type`* 是 `no-repeat`、`repeat-x` 和 `repeat-y` 中的一种（通常设为 `no-repeat`），*`horizontal`* 和 *`vertical`* 值表示列表项目中背景图像的位置（如图 15.3.2 所示）。

(4) 输入 `padding-left:` *`value`*`;`，这里的 *`value`* 应不小于背景图像的宽度，以防列表项目的内容覆盖到定制标记的上面。

```
ul {
    /* 取消默认标记 */
    list-style: none;

    /* 删除列表项的缩进 */
    margin-left: 0;
    padding-left: 0;
}

li {
    /* 显示定制的标记 */
    background: url(../img/checkmark.png)
    → no-repeat 0 0;
}
```

图 15.3.2　下面会用三步法展示如何使用定制的标记，这样读者会对各种 CSS 属性如何影响布局一目了然。首先，清除默认的标记（这样就看不到符号和箭头了），如果想删除列表项目的缩进，应该将 `margin-left` 和 `padding-left` 都设为 0（参见倒数第二条提示）。应用于列表项的定制标记的 URL 看起来不一样（参见第一条提示），通常使用 PNG，但 GIF 或 JPEG 也行

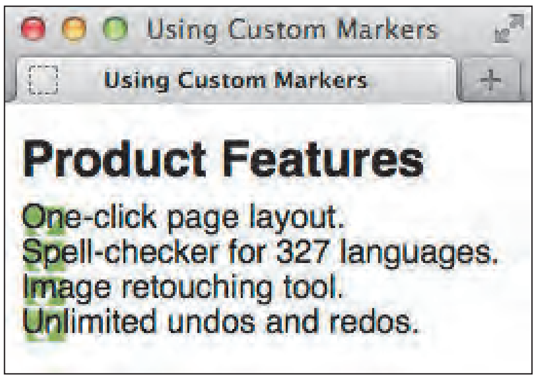


图 15.3.3　默认的圆点被核对图像取代了，列表项之前的缩进被移除了，但是文本叠加到了新的标记上，那么要不要针对列表添加左内边距，从而让标记和列表项之间有点空间呢？让我们来试试，请看图 15.3.4

```
ul {
    list-style: none;
    margin-left: 0;
    padding-left: 30px;
    outline: 2px solid red;
}

li {
    background: url(../img/checkmark.png)
    → no-repeat 0 0;
    outline: 1px solid blue;
}
```

图 15.3.4　将 margin-left 设为 0 后，就可以针对所有浏览器使用 padding-left 控制列表项目的缩进了。我对 ul 和每个 li 都添加了一个临时的边框，从而可以清晰地看到应用内边距的位置（参见图 15.3.5）

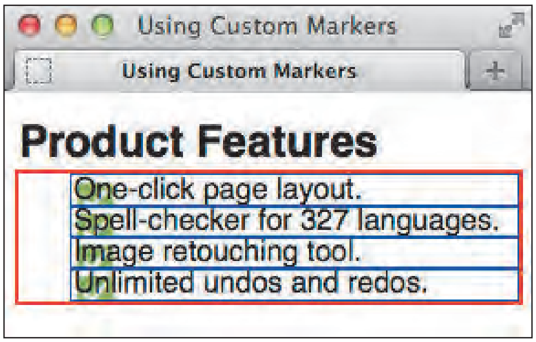


图 15.3.5　这样做让列表项目有了缩进，但文字依然与标记重叠了，这是因为我们仅为列表本身（ul）添加了左内边距，但没有对包含标记的列表项目添加左内边距。下面将修复这个问题（参见图 15.3.6）

提 示　注意，相对背景 URL 是相对于样式表的位置，而不是相对于网页的位置（参见图 15.3.2），相关内容参见 10.10 节。url 和前括号之间不应有空格，包围 URL 的引号是可选的。

```
ul {
    list-style: none;
    margin-left: 0;
    padding-left: 0;
}

li {
    /* 让图像在列表项目内显示的位置稍微往
       下一些 */
    background: url(../img/checkmark.png)
    → no-repeat 0 .1em;

    /* 让行高变大，从而可以容纳整个标记 */
    line-height: 1.8;

    /* 将文本推开，为标记腾出位置 */
    padding-left: 1.75em;
}
```

图 15.3.6　将 ul 的左边距重新设为 0，列表会再次向左对齐。为每个 li 设置左内边距可以将所有标记图像显示出来，增大 line-height 则可以保证图像底部不会被截断。此外，这里还让标记向下移动了一些，从而可以与文本更好地对齐（如图 15.3.7 所示）

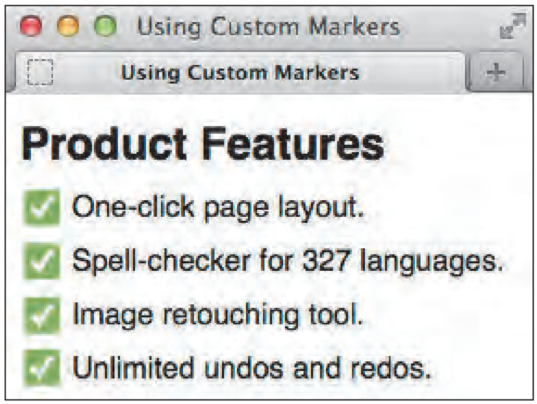


图 15.3.7　好多了

提 示　默认标记在默认情况下位于列表项目的外面，而定制标记显示在列表项目的里面，因为定制标记是为列表项目本身应用的背景图像。二者的对比参见 15.1 节的图 15.1.5 ～图 15.1.7。那些内容还演示了如何实现本节图 15.3.7 所示的减小标题下面的间距和格式化文本的样式。

15

提示　如果想为列表中的某一个或几个项目应用定制的标记，可以为其添加一个类，再为这个类定义样式规则。

提示　大多数浏览器是通过 padding-left 为列表设置默认缩进的，但旧浏览器（如 IE8 之前版本）则是通过 margin-left 设置的。这就是要在使用 padding-left 获得统一样式结果之前需要设置 margin-left: 0; 的原因。现在，尽管通常不必为 IE7 及更旧的浏览器考虑了（对于大多数国家来说），但添加 margin-left: 0; 也没有任何坏处。

提示　另一种显示定制标记的方法是使用 list-style-image 属性。例如，li { list-style-image: url(image.png); }。不过，使用这种方法很难达到预期的效果，因为不同浏览器对它的显示效果并不一致。并且相比前面展示的背景图像方法，开发者更难控制图像标记的位置。

15.4　选择列表的起始编号

你可能希望列表的编号从默认值 1 以外的某个数字开始，如图 15.4.1 和图 15.4.2 所示。

1. 设置整个列表编号方案的初始值

在 ol 开始标签中输入 start="*n*"，这里的 *n* 表示列表的初始值。

2. 修改有序列表中某列表项目的编号

在目标 li 项目内输入 value="*n*"，这里的 *n* 代表该列表项目的值。

提示　如果使用 start 和 type，请始终使用数字。即便决定使用字母或罗马数字对列表进行编号（参见 15.2 节），也应使用数字。通过 CSS 或者 type 属性会显示预期的标记。

```
...
<body>

<h1 class="hdg">Changing a light bulb (with a
→ few steps missing)</h1>

<ol start="2">
    <li>Unscrew the old bulb.</li>
    <li value="5">Screw in the new bulb.
    → </li>
    <li>Plug in the lamp and turn it on!
    → </li>
</ol>

</body>
</html>
```

图 15.4.1　在这个例子中，我略去了列表中的一些步骤，不过仍希望余下的步骤保持原有的编号。因此，整个列表从 2 开始编号（使用 start="2"），并将第二个项目的值设为 5（使用 value="5"）。这两个属性都是可选的，也不一定要像这样同时使用

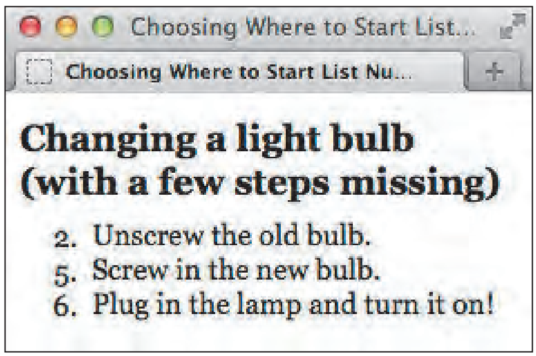


图 15.4.2　注意，不仅第一个和第二个项目是按我们指定的规则进行编号的，连第三个项目（Plug in the lamp and turn it on!）也受到了影响（我为页面添加了一些样式，但并没有将 CSS 写在这里。我将 body 字体设为 Georgia，对 h1 应用了 class="hdg"，如图 15.4.1 所示，并应用了 15.1 节中图 15.1.6 中的 .hdg 样式规则）

提示　value 属性的值会覆盖 start 属性的值。

提示　使用 value 属性对某列表项目的编号进行修改后，后续的列表项目也会相应地重新编号。

提示　如果需要在有序列表中指定两个或两个以上位置相同的项目，使用 value 是非常方便的。以表示公路赛前五名选手的列表为例，通常该列表会显示为 1、2、3、4、5，但如果有并列第 2 名，则可以将第三个项目写成 <li value="2">，这时，列表将显示为 1、2、2、3、4。

提示　列表可以包含一个以上的带 value 属性的 li。

15.5　控制标记的位置

默认情况下，列表会从（其父元素的）左侧进行缩进。标记可以位于整个文本块的外面（这是默认情况），如图 15.5.1 所示，也可以缩在文本块内（称为缩排），如图 15.5.2 和图 15.5.3 所示。

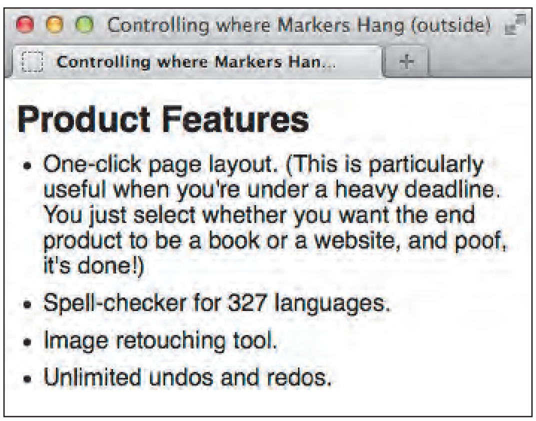


图 15.5.1　这说明了在默认情况下浏览器呈现列表项目文本及标记之间相对位置的方式。标记位于内容的外边

```
ul {
    list-style-position: inside;
}
```

图 15.5.2　将 list-style-position 设置为 inside 可以改变显示方式

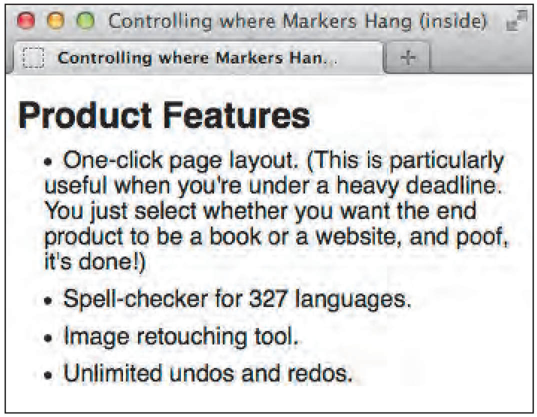


图 15.5.3　每行的标记都是从列表项目的左侧外边距开始的，而不是在整体内容之外

控制标记位置的步骤

(1) 在目标列表或列表项目的样式表规则中，输入 list-style-position:。

(2) 输入 inside 让标记缩在文本块内（参见图 15.5.2），或者输入 outside 让标记显示在列表项目文本的左边（这是默认的设置）。

提示　不需要指定 list-style-position: outside;，因为 outside 是默认值，除非需要覆盖其他位置设置的 list-style-position: inside;。

提示　list-style-position 可应用于 ul（参见图 15.5.2）、ol 或 li。结果是一样的，除非针对特殊的 li（如某一类的 li）设置不同的样式，从而在一个列表中同时拥有内部标记和外部标记。

提示　如果列表项目中的文本显得较为拥挤，就像图 15.5.1 和图 15.5.3 中的那样，则可以通过设置 line-height 增大文本行间距。例如，可以设置 li { line-height: 1.3; }。不要将行高的设置与列表项目间距（由对 li 设置的 margin-top 或 margin-bottom 控制）弄混。例如，li { margin-bottom: .5em; }。

提示　如何减少列表项前面的缩进和标题下面的间距，以及如何格式化文本参见图 15.1.5 至图 15.1.7。

提示　list-style-position 属性是继承的。

15.6　同时设置所有的列表样式属性

同 background、border、font、outline 等属性一样，CSS 也为 list-style 提供了简写形式，参见图 15.6.1 和图 15.6.2。它将 list-style-type、list-style-position 和较少使用的 list-style-image 放到了一条属性中。

```
ul {
    list-style: circle inside;
}

/* 将ol作为选择器为无序列表添加样式，有序列表
   和无序列表都可以使用li */
```

图 15.6.1　使用这条样式规则等价于将 list-style-type 设为 circle，并将 list-style-position 设为 inside，只是这样写更为简洁。如果想在简写属性中指定 list-style-image，这个例子可以写作 ul { list-style: url(arrow-right.png) circle inside; }。但第 15.3 节提到过，最好对 li 应用背景图像，而不是使用 list-style-image

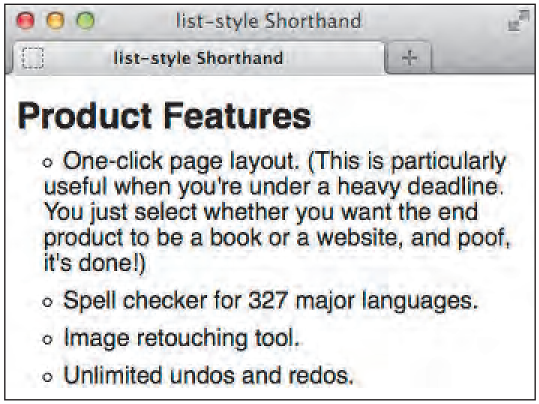


图 15.6.2　代码的实现结果与图 15.5.3 相同，只是将标记改成了圆圈

同时设置所有的列表样式属性的步骤

(1) 输入 list-style:。

(2) 如果需要，指定应出现在列表项目旁边的标记类型（参见 15.2 节）。

(3) 如果需要，指定标记应悬挂在列表段落之外或缩在文本块内（参见 15.5 节）。

(4) 如果需要，指定列表项目所用的定制图像标记（参见 15.3 节最后一条提示）。

提示　可以指定三个 list-style 属性，也可以指定其中的任意一个，指定多个属性时顺序任意。图 15.6.1 中指定了两个。任何没有显式地指定的属性都会设为其默认值（list-style-type 的默认值为 disc，list-style-image 的默认值为 none，list-style-position 的默认值为 outside）。

提示　关于 list-style，或许最为常见的用法是使用 list-style: none 取消标记。

提示　跟 list-style-type、list-style-position 和 list-style-image 属性一样，list-style 属性是继承的。这也是我们可以将它应用到父元素 ol 或 ul 上的原因。

15.7 设置嵌套列表的样式

可以在一个列表中插入另一个列表，里面的列表就称为**嵌套**列表。对于有序列表和无序列表均可创建嵌套列表（混合在一起或单独嵌套）。此外，还有另外一种嵌套列表，有关示例参见 15.8 节。

如果要按有序列表的结构创建大纲（可能需要好几级的列表项目，如图 15.7.1、图 15.7.2 和图 15.7.3 所示），或者需要按无序列表的结构创建带子菜单的导航（如图 15.7.4 和图 15.7.5 所示，更多细节参见补充材料“使用嵌套列表创建下拉式导航”），就会发现嵌套列表很有用。可以通过很多方式为嵌套列表设置样式，就像示例所展示的那样。

为嵌套列表设置样式的步骤

(1) 要设置最外层列表的样式，输入 *toplevel li* {*style_rules*}，其中，*toplevel* 是最外层列表的类型（如 ol、ul），*style_rules* 是要应用的样式。

(2) 对于第二级列表，输入 *toplevel 2ndlevel* {*style_rules*}，其中，*toplevel* 对应于第 (1) 步中的 *toplevel*，*2ndlevel* 则是第二级列表的类型，而 *style_rules* 是它需要应用的样式。

(3) 对于第三级列表，输入 *toplevel 2ndlevel 3rdlevel* {*style_rules*}，其中，*toplevel* 和 *2ndlevel* 对应于第 (1) 步和第 (2) 步中的 *toplevel* 和 *2ndlevel*，*3rdlevel* 则是第三级列表的类型。而 *style_rules* 则是它需要应用的样式。

(4) 对于要设置样式的每个嵌套列表，继续以上述方法进行设置。

可以在每个选择器末尾包含 li，从而直接定位列表项目，例如，第 (3) 步中的代码可以写作 *toplevel 2ndlevel 3rdlevel* li {*style_rules*}，图 15.7.2 中的示例代码中有这两种例子。

```
...
<body>
<h1 class="hdg">The Great American Novel</h1>
<ol>
    <li>Introduction
      <ol>
        <li>Boy's childhood</li>
        <li>Girl's childhood</li>
      </ol>
    </li>
    <li>Development
      <ol>
        <li>Boy meets Girl</li>
        <li>Boy and Girl fall in love
        ➝ </li>
        <li>Boy and Girl have fight
        ➝ </li>
      </ol>
    </li>
    <li>Climax
      <ol>
        <li>Boy gives Girl ultimatum
          <ol>
            <li>Girl can't believe
            ➝ her ears</li>
            <li>Boy is indignant at
            ➝ Girl's indignance</li>
          </ol>
        </li>
        <li>Girl tells Boy to get
        lost
        ➝ </li>
      </ol>
    </li>
    <li>Denouement</li>
    <li>Epilogue</li>
</ol>
</body>
</html>
```

图 15.7.1 注意每个嵌套的 ol 都包含在其父元素的开始标签 <li> 和结束标签 之间。这里有四个嵌套列表，一个位于 Introduction 列表项目内，一个位于 Development 列表项目内，一个位于 Climax 列表项目内，还有一个突出显示的粗体字列表位于 Boy gives Girl ultimatum 列表项目内（该列表项目又位于 Climax 列表项目内）

```
ol {
    list-style-type: upper-roman;
}

ol ol {
    list-style-type: upper-alpha;
}

ol ol ol {
    list-style-type: decimal;
}

ol li {
    font-size: .875em;
}

li li {
    font-size: 1em; /* 防止文本不断缩小 */
}
```

图 15.7.2　可以对每一级嵌套列表单独进行格式化。如果要对列表中的文本使用 em 或百分数设置字体大小，就需要添加 li li {font-size: 1em; }（或者将 1em 替换为 100%），以防止嵌套列表不断缩小而导致难以辨认（参见最后一条提示）

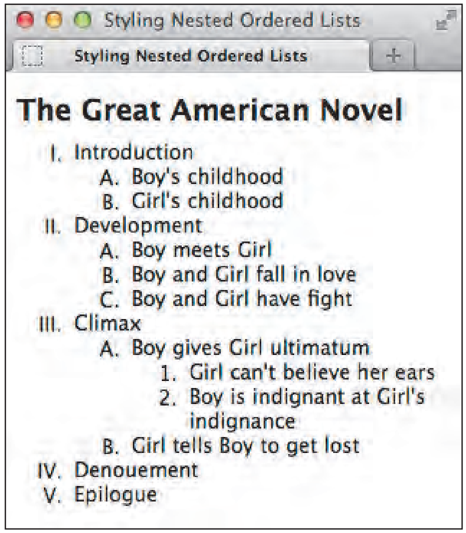


图 15.7.3　第一级列表（ol）使用大写罗马数字，第二级列表（ol ol）使用大写字母，第三级列表（ol ol ol）使用阿拉伯数字

```
...
<body>
<nav role="navigation">
    <ul class="nav">
       <li><a href="/">Home</a></li>
       <li><a href="/products/">Products</a>
          <ul class="subnav">
             <li><a href="/products/phones.html">Phones</a></li>
             <li><a href="/products/accessories.html">Accessories</a></li>
          </ul>
       </li>
       <li><a href="/support/">Support</a>
          <ul class="subnav">
             <li><a href="/support/forum/">Community Forum</a></li>
             <li><a href="/support/contact-us.html">Contact Us</a></li>
             <li><a href="/support/how-to-guides.html">How-to Guides</a></li>
          </ul>
       </li>
       <li><a href="/about-us/">About Us</a></li>
    </ul><!-- end .nav -->
</nav>
...
</body>
</html>
```

图 15.7.4　这是另一个嵌套列表的例子。在这个例子中，使用无序列表构建导航菜单（带有 class="nav"），同时使用两个嵌套的无序列表构建子菜单（每个都带有 class="subnav"）。通过一些 CSS，可以让导航水平排列，同时让子菜单在默认情况下隐藏起来，并在访问者激活它们时显示出来，如图 15.7.5 所示

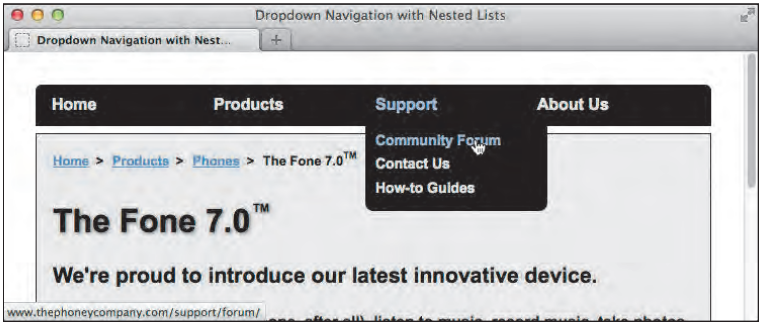


图 15.7.5 Products 和 Support 列表项目都包含嵌套在 ul 中的子菜单，不过，由于应用了一些 CSS，这两个子菜单在默认情况下并不显示出来。在这个例子中，Support 的子菜单显示出来了，这是因为鼠标停留在包含 Support 链接及子菜单嵌套列表（参见图 15.7.4）的 li 上的缘故。这个截图也显示了一个面包屑导航的例子（位于黑色横条的下方）。该面包屑导航使用有序列表进行标记，用于指示当前页面在整个网站层级中的位置。其中，每个列表项目都是一个链接，但最后一个不是链接，因为用户所在的页面正是 The Fone 7.0。完整的 CSS 见本书配套网站

15

提示 选择器应反映文档中嵌套列表的类型，例如，可能需要使用 `ul ul ol` 这样的选择器，或者在这个例子中使用 `ul ul ol li` 直接定位项目列表。

提示 除了上述方法以外，也可以为每个嵌套列表添加类名，再设置对应的样式。图 15.7.2 中设置的方法允许我们可以在不改变 HTML 的情况下控制样式。

提示 无论嵌套的层级是哪一级，有序列表在默认情况下总是使用阿拉伯数字（1、2、3）。可以使用 `list-style-type` 指定其他的编号方案（参见 15.4 节）。根据 *The Chicago Manual of Style* 一书的建议，正确的列表嵌套次序为 I（罗马数字）、A、1、a（此后交替使用 1 和 a 编号方案）。

提示 默认情况下，对于无序列表，第一级列表使用圆点符号，下一级列表使用空心圆圈符号，第三级及后续各级列表使用方块符号。类似地，可以使用 `list-style-type` 指定希望使用的符号类型（参见 15.2 节）。

提示 由于列表项目（li 元素）可以嵌套在其他列表项目内，因此在使用相对值指定字体大小时需要留心。如果使用像 `li {font-size: .75em; }` 这样的样式规则，那么最外层列表项目的字体大小便是其父元素的 75%，因此如果该父元素的字体大小为默认的 16 像素，那么最外层列表项目就是 12 像素大小，这还算好。但是，第一个**嵌套**列表项目的字体大小会是其父元素（第一个列表项目，12 像素大小）的 75%，因此只有 9 像素。每一级都会迅速让情况变得更糟糕。一种解决方案是添加 `li li {font-size: 1em;}`，如图 15.7.2 所示（或者将 1em 替换为 100%）。这样，嵌套列表项目就总是与顶级列表项目一样大，如图 15.7.3 所示。（由 Eric Meyer 提供，www.meyerweb.com。）

使用嵌套列表创建下拉式导航

嵌套列表的一种使用场合是创建下拉式（或飞出式）导航菜单（如图 15.7.4 所示）。通过使用 CSS 为导航添加一些样式，可以让每个子菜单仅在访问者鼠标停留在其父元素的列表项目上时显示出来（如图 15.7.5 所示），并在访问者将鼠标移走时隐藏。

可以通过多种方法实现这一效果，但这些方法总离不开在选择器中使用 :hover 伪类以显示子菜单。下面演示了这样一种在默认情况下隐藏嵌套列表，并在访问者鼠标停留时显示该列表的方法：

```
/* 子菜单的默认状态 */
.nav .subnav {
    left: -999em; /* 将子菜单移出屏幕 */
    position: absolute;
    z-index: 1000;
}

/* 当鼠标停留在父元素li上时子菜单的状态 */
.nav li:hover .subnav {
    left: auto; /* 让子菜单回到自然状态 */
}
```

对应的 HTML 如图 15.7.4 所示。要实现水平排布，移除列表项目的符号，以及对外观进行调整，都需要使用更多的 CSS。图 15.7.5 中所示页面的完整 HTML 和 CSS 代码见本书配套网站 www.htmlcssvqs.com/8ed/15/dropdown-nav。我还在代码中添加了一些注释。

可以使用类似的方法创建带飞出式子菜单的垂直导航（子菜单显示在导航的侧边）。

15.8　创建描述列表

HTML 提供了专门用于描述成组出现的名称（术语）及其值之间关联的列表类型。这种类型在 HTML5 中称为**描述列表**（description list），而在 HTML 的早期版本中则称为定义列表。

根据 HTML5 规范，“由名称及其值构成的组合可以是术语和定义、元数据主题和值、问题和答案，以及任何其他的名 – 值组。”每个列表都包含在 dl 中，其中的每个名 – 值组都有一个或多个 dt 元素（名称或术语）以及一个或多个 dd 元素（它们的值）。图 15.8.1 展示了一个基本的描述列表示例。除了使用一个简单的样式规则（如图 15.8.2 所示）应用粗体以外，所有的样式都是默认的（如图 15.8.3 所示）。

```
...
<body>
<h1>List of Horror Movie Legends</h1>

<dl>
    <dt>Boris Karloff</dt>
    <dd>Best known for his role in <cite>Frankenstein</cite> and related horror films, this
    → scaremaster's real name was William Henry Pratt.</dd>

    <dt>Christopher Lee</dt>
    <dd>Lee took a bite out of audiences as Dracula in multiple Hammer horror classics.</dd>

    ... [更多恐怖传说] ...
</dl>

</body>
</html>
```


图 15.8.1　这是最基本的描述列表，每个名 – 值组都由一个 dt 和一个 dd 构成。每个组之间的空行仅仅是为了提高代码的可读性而添加的。这些空行不是必需的，它们不改变内容的含义，也不影响内容的呈现

```
dt {
    font-weight: bold;
}
```

图 15.8.2　你可能想为 dt 元素中的术语添加一些格式，从而让它们突出显示（参见图 15.8.3）



图 15.8.3　在默认情况下，名称（dt）向左对齐，而值（dd）则有缩进。由于图 15.8.2 中设置的简单样式，名称以粗体显示。如果没有设置该样式，它们将以常规文本显示

以下都是 dl 元素内的 dt 和 dd 元素的组合，这些安排都是有效的。

- ❑ 一个 dt 同一个 dd（参见图 15.8.1，同时参见图 15.8.7 中 Cast 下嵌套描述列表中的 Director）。这是最常见的情形。
- ❑ 一个 dt 同多个 dd 元素，参见图 15.8.7 中的 Writers。
- ❑ 多个 dt 元素同一个 dd，如图 15.8.4 所示。（样式设置的示例如图 15.8.5 和图 15.8.6 所示。）
- ❑ 多个 dt 元素同多个 dd 元素。如果图 15.8.4 中的 bogeyman/boogeyman 有多个定义，就是这种情形。

可以使用 dfn 元素包围 dt 中的名称，指出该列表是用于定义术语的，如在术语表中，如图 15.8.4 所示。（关于 dfn，参见 4.9 节。）

可以对描述列表进行嵌套（如图 15.8.7 所示），并通过 CSS 对它们添加所需的样式（如图 15.8.8 所示）。在默认情况下，如果一个 dl 嵌套在另一个 dl 中，它会自动进行缩进，如图 15.8.9 所示（当然也可以通过 CSS 对此进行修改）。

```
...
<body>

<h1>Defining words with multiple spellings</h1>

<dl>
    <dt><dfn>bogeyman</dfn>, n.</dt>
    <dt><dfn>boogeyman</dfn>, n.</dt>
    <dd>A mythical creature that lurks under
    → the beds of small children.</dd>

    <dt><dfn lang="en-gb">aluminium
    → </dfn>, n.</dt>
    <dt><dfn>aluminum</dfn>, n.</dt>
    <dd>...</dd>
</dl>
</body>
</html>
```

图 15.8.4　在这个例子中，每个名–值组都包含多个 dt 和一个 dd，因为这些术语有一个以上的拼写方式，而它们的定义是相同的

```
dd + dt {
    margin-top: 1em;
}
```

图 15.8.5　这会在每个名–值组之间添加比默认设置更多的空间

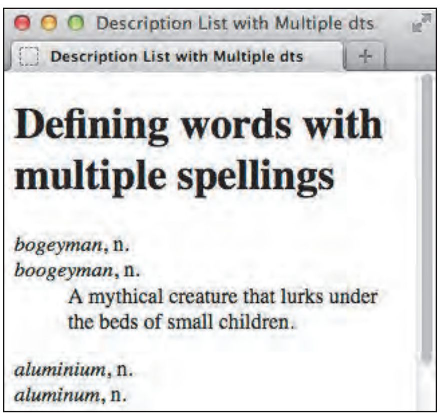


图 15.8.6　现在可以清楚地看到各组描述之间的间隔了。由于 *aluminium, n.* 所在的 dt 紧跟在上一名–值组中的 dd 后面，因此图 15.8.5 中的样式规则发挥了作用

创建描述列表的步骤

(1) 输入 <dl>。

(2) 输入 <dt>。

(3) 输入需要描述或定义的单词或短语，包括任何额外的语义元素（如 dfn）。

(4) 输入 </dt> 以完成名–值组中的名称部分。

(5) 如果名–值组中有一个以上的名称或术语，可根据需要重复第 (2) 步至第 (4) 步（参见图 15.8.4）。

```
...
<body>

<h1>Credits for <cite>Amélie</cite></h1>

<dl>
    <dt>Director</dt>
    <dd>Jean-Pierre Jeunet</dd>

    <dt>Writers</dt>
    <dd>Guillaume Laurant (story,
    → screenplay)</dd>
    <dd>Jean-Pierre Jeunet (story)</dd>

    <dt>Cast</dt>
    <dd>
      <!-- 开始嵌套列表 -->
      <dl>
        <dt>Audrey Tautou</dt>
        <dd>Amélie Poulain</dd>

        <dt>Mathieu Kassovitz</dt>
        <dd>Nino Quincampoix</dd>

        ... [rest of Cast] ...
      </dl>
      <!-- 结束嵌套列表 -->
    </dd>

    ... [其他致谢名单] ...
</dl>

</body>
</html>
```

图 15.8.7　这是一个描述列表的例子，它描述了一部电影的导演、编剧和演员表，其中的演员表是由演员姓名及其角色构成的嵌套描述列表。可以根据需要对嵌套的列表设置不同的样式（如图 15.8.8 所示）

(6) 输入 <dd>。

(7) 输入第 (3) 步中输入的术语或名称的描述。

(8) 输入 </dd> 以完成名 – 值组中的描述（值）部分。

(9) 如果名 – 值组中有一个以上的定义值，可根据需要重复第 (6) 步至第 (8) 步（参见图 15.8.7 中的 Writers）。

(10) 对每个由术语和描述构成的组，重复第 (2) 步至第 (9) 步。

(11) 输入 </dl> 以完成整个描述列表。

```
body {
    font-family: Verdana, Geneva, sans-serif;
}

h1 {
    font-size: 1.75em;
}

dt {
    font-weight: bold;
    text-transform: uppercase;
}

/* 为位于另一个dl中的任意dl的dt设置样式 */
dl dl dt {
    text-transform: none;
}

dd + dt {
    margin-top: 1em;
}
```

图 15.8.8　我想对主要列表中的术语和嵌套列表中的术语进行区分，因此我对 dt 元素使用了大写字母样式，再将位于嵌套 dl 中的 dt 元素重新设为常规样式（使用 text-transform: none; 声明）。不过，注意所有的术语均以粗体显示（如图 15.8.9 所示），这是因为第一条样式规则中的声明适用于所有的 dt 元素，同时并未在嵌套列表的样式中清除这一样式

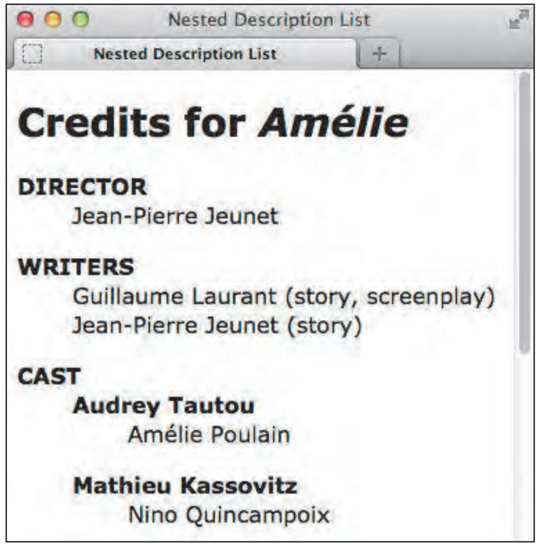


图 15.8.9　在默认情况下，当一个 dl 嵌套在另一个 dl 中时，嵌套的列表会自动进行缩进。根据图 15.8.8 中的样式，第一级 dt 元素使用大写字母，而嵌套列表中的 dt 元素则使用常规样式。所有的 dt 元素均以粗体显示

提 示　对于描述（值），浏览器通常会在其术语（名称）下面新的一行对其进行缩进（参见图 15.8.3）。可以通过自定义 dd 元素的 margin-left 值改变缩进。如 dd { margin-left: 0; } 会将描述跟术语左对齐。

提 示　你应该已经在示例（图 15.8.1、图 15.8.4、图 15.8.7）中看到，不必（更确切地说，不应该）使用 p 元素对 dd 元素中的单个文本段落进行标记。不过，如果**单个**描述是由一个以上的段落构成的，就**应该**在一个 dd 元素中使用多个 p 元素对其进行标记，而不是将每个段落（不使用 p 元素）放入单独的 dd。

第 16 章

表单

16

本章内容

- HTML5 对表单的改进
- 创建表单
- 处理表单
- 对表单元素进行组织
- 创建文本框
- 为表单组件添加标签
- 创建密码框
- 创建电子邮件框、搜索框、电话框和 URL 框
- 创建单选按钮
- 创建复选框
- 创建文本区域
- 创建选择框
- 让访问者上传文件
- 创建隐藏字段
- 创建提交按钮
- 禁用表单元素
- 根据状态为表单设置样式

到目前为止，你学到的所有 HTML 都是用于帮助你将自己的想法告诉访问者的。在本章中，你将学习如何创建表单，与访问者进行交流。

表单有两个基本组成部分：访问者在页面上可以看见并填写的控件、标签和按钮的集合；以及用于获取信息并将其转化为可以读取或计算的格式的处理脚本。本章主要关注第一部分：创建表单。本书配套网站上提供了许多示例脚本，参见 www.htmlcssvqs.com/8ed/form-scripts。

构造表单的字段和按钮很直观，同创建网页的其他部分是相似的。基本的表单字段类型包括文本框、单选按钮、复选框、下拉菜单、更大的文本区域。如果你曾在网上购过物，曾加入过社交网络，或撰写过基于 Web 的电子邮件，那么你会很熟悉即将在这里学到的表单元素。我还会演示如何使用 CSS 设置表单的样式。

16.1　HTML5 对表单的改进

如果你没有接触过表单，最好先跳过本节，待熟悉本章其余内容之后再回过头来阅读本节。

HTML5 的一个重要的特性就是对表单的改进。过去，我们常常需要花费很多额外的时间，编写 JavaScript 以增强表单行为——例如，要求访问者提交表单之前必须填写某个字段。HTML5 通过引入新的表单元素、输入类型和属性，以及内置的对必填字段、电子邮件地址、URL 以及定制模式的验证，让这一切变得很轻松。这些特性不光帮助了设计人员和开发人员，也让网站访问者的体验有了很大的提升。

更妙的是，不支持这些新特性的旧版浏

览器也不会出太大的问题。它们会直接忽略这些无法理解的属性，表单里的输入框也会正常显示。如果你希望它们拥有 HTML5 行为，可以使用 JavaScript 填补空白，就像过去那样（参见最后一条提示）。

表 16.1.1 和表 16.1.2 总结了 HTML5 大部分与表单相关的特性，并指出了如何获取更多信息。你将看到，我们会将重点放在最常使用的那些特性上面。

表 16.1.1　输入和元素

输入或者元素	简略代码	更多信息
电子邮件框	<input type="email">	参见 16.8 节
搜索框	<input type="search">	
电话框	<input type="tel">	
URL 框	<input type="url">	
以下元素得到了部分浏览器的支持		
日期	<input type="date">	更多信息参见：www.wufoo.com/html5 浏览器支持： caniuse.com/#feat=input-datetime caniuse.com/#feat=input-number caniuse.com/#feat=input-range
数字	<input type="number">	
范围	<input type="range">	
数据列表	<input type="text" name="favfruit" → list="fruit" /> <datalist id="fruit"> <option>Grapes</option> <option>Pears</option> <option>Kiwi</option> </datalist>	更多信息参见：www.wufoo.com/html5
下面的输入或者元素得到的支持很少，W3C 指出它们在 2014 年定案之时很可能不会列入 HTML5		
颜色	<input type="color" />	www.w3.org/html/wg/wiki/HTML5.0AtRiskFeatures
全局日期和时间	<input type="datetime" />	
局部日期和时间	<input type="datetime-local" />	更多信息参见：www.wufoo.com/html5 浏览器支持： caniuse.com/#feat=input-color caniuse.com/#feat=input-datetime
月	<input type="month" />	
时间	<input type="time" />	
周	<input type="week" />	
输出	<output></output>	

表 16.1.2　属性

属　性	总　结	更多信息
accept	限制用户可上传文件的类型	www.wufoo.com/html5
autocomplete	如果对 form 元素或特定的字段添加 autocomplete="off"，就会关闭浏览器的对该表单或该字段的自动填写功能。默认值为 on	16.5 节
autofocus	页面加载后将焦点放到该字段	16.5 节
multiple	允许输入多个电子邮件地址，或者上传多个文件	16.8 节
list	将 datalist 与 input 联系起来	www.wufoo.com/html5
maxlength	指定 textarea 的最大字符数（在 HTML5 之前的文本框就支持该特性）	16.11 节
pattern	定义一个用户所输入的文本在提交之前必须遵循的模式	16.8 节
placeholder	指定一个出现在文本框中的提示文本，用户开始输入后该文本消失	16.5 节
required	需要访问者在提交表单之前必须完成该字段	16.5 节
formnovalidate	关闭 HTML5 的自动验证功能。应用于提交按钮	16.15 节
novalidate	关闭 HTML5 的自动验证功能。应用于表单元素	16.2 节

提示　对于浏览器支持信息，caniuse.com 上的信息通常比 www.wufoo.com/html5 上的更新一些，不过后者仍然是有关 HTML5 表单信息的一个重要资源。

提示　那些有可能不会成为 HTML5 一部分的表单元素，也有可能被收录到 HTML 5.1 的最终版本（预计 2016 年发布）。

提示　Ryan Seddon 的 H5F（https://github.com/ryanseddon/H5F）可以为旧的浏览器提供模仿 HTML5 表单行为的 JavaScript 方案。

16.2　创建表单

每个表单都以 form 开始标签开始，以 form 结束标签结束。两个标签之间是组成表单的说明标签、控件和按钮，如图 16.2.1 所示。（注意，我会交替使用**控件**（control）和**字段**（field）两个词。）每个控件都有一个 name 属性，用于在提交表单时对数据进行识别。访问者通过你提供的提交按钮提交表单——触发提交按钮时，他们填写的数据就会发送至服务器上处理数据的脚本。

了解了表单的全貌之后，不妨仔细看看其中的细节。form 开始标签可以有一些属性，其中最重要的就是 action 和 method（如图 16.2.1 所示）。

将 action 属性的值设为访问者提交表单时服务器上对数据进行处理的脚本的 URL。例如，action="save-info.php"。

method 属性的值要么是 get，要么是 post。大多数情况下都可以使用 post，不过每种方法都有其用途，了解其用途有助于理解它们。更多细节参见本节的补充材料。

```
...
<body>
<h1>Create a New Account</h1>
<form method="post" action="show-data.php">
    <!-- 各种表单元素 -->
    <fieldset>
       <h2 class="hdr-account">Account</h2>

       <div class="fields">
          <p class="row">
             <label for="first-name">First Name:</label>
             <input type="text" id="first-name" name="first_name" class="field-large" />
          </p>
          <p class="row">
             <label for="last-name">Last Name:</label>
             <input type="text" id="last-name" name="last_name" class="field-large" />
          </p>
          ...
       </div>
    </fieldset>
    ... 更多表单元素 ...

    <!-- 提交按钮 -->
    <input type="submit" value="Create Account" class="btn" />
</form>
</body>
</html>
```

图 16.2.1　每个表单包括 form 元素自身、访问者输入信息的其他表单元素以及用于将收集到的信息发送给服务器的提交按钮

创建表单的步骤

(1) 输入 <form method="*formmethod*"，这里的 formmethod 是 get 或者 post。

(2) 输入 action="*script.url*">，这里的 *script.url* 是提交表单时要运行的脚本在服务器上的位置。

(3) 根据从 16.5 节开始讲解的知识，创建表单的内容（包括一个提交按钮）。

(4) 输入 </form> 以结束表单。

提示　可以使用 CSS 布局表单元素，参见图 16.2.2。此处演示的这个贯穿本章的表单示例显示在浏览器中如图 16.2.3 所示。

提示　图 16.2.1 和图 16.2.2 中完整的 HTML 和 CSS 代码可以从本书配套网站下载（www.htmlcssvqs.com/8ed/16），其中部分代码会贯穿本章。此外，配套网站上还有表单在 action 属性中引用的 show-data.php 脚本。学习本章过程中，你可以随意使用它来测试你的表单。需要注意的是，除非你的服务器安装了 PHP，否则这个脚本不工作。

提示　可以对 form 元素应用 novalidate 属性，关闭表单的 HTML5 验证特性（本章后面会演示这一特性）。例如 <form method="post" action="show-data.php" novalidate>。

```
fieldset {
    background-color: #f1f1f1;
    border: none;
    border-radius: 2px;
    margin-bottom: 12px;
    overflow: hidden;
    padding: 0 .625em; /* 10px */
}

.fields {
    background-color: #fff;
    border: 1px solid #eaeaea;
    margin: .75em; /* 12px */
    padding: .75em;
}

.fields .row {
    margin: 0.5em 0;
}

label {
    cursor: pointer;
    display: inline-block;
    padding: 3px 6px;
    text-align: right;
    width: 150px;
    vertical-align: top;
}

input, select, button {
    font-size: inherit;
}

/* 各种表单字段的宽度 */
.field-small {
    width: 75px;
}

.field-medium {
    width: 150px;
}

.field-large {
    width: 250px;
}
```

图 16.2.2　这是用于对表单进行格式化的样式表的一部分。完整的样式表见本书的配套网站

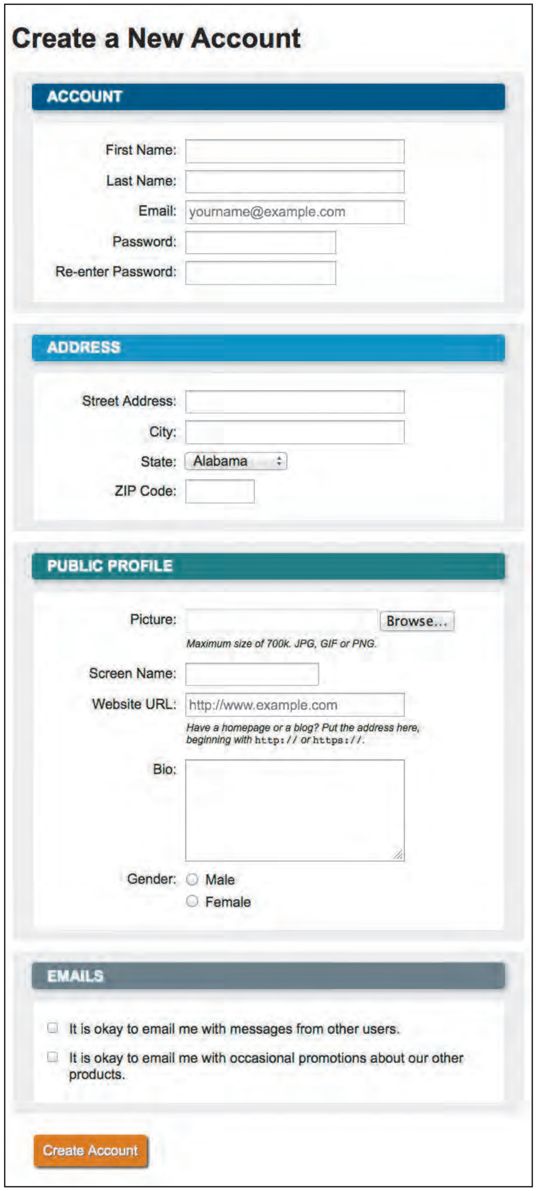


图 16.2.3　这是本章讨论的完整的 New Account 表单

method="get" 与 method="post" 的区别

前面提到，form 元素的 method 属性要么是 get，要么是 post。

如果对表单使用 method="get"，那么表单提交后，表单中的数据会显示在浏览器的地址栏里。通常，如果你希望表单提交后从服务器得到信息，就使用 get。例如，大多数搜索引擎都会在搜索表单中使用 get——输入 Kermit meets Yoda，提交表单，搜索引擎会得到搜索结果。由于数据出现在 URL 中，因此用户可以保存搜索查询，或者将查询发给朋友。

如果对表单使用 method="post"，那么提交表单后，表单中的数据不会显示在浏览器的地址栏里，这样更为安全。同时，比起 get，使用 post 可以向服务器发送更多的数据。通常，post 用于向服务器存入数据，而非获取数据。因此，如果需要在数据库中保存、添加和删除数据，就应选择 post。例如，电子商务网站使用 post 保存信用卡、邮件地址以及其他用户输入的信息。

通常，如果不确定使用哪一种，就使用 post，这样数据不会暴露在 URL 中。

16.3　处理表单

表单从访问者那里收集信息，脚本则对这些信息进行处理。脚本可以将信息记录到服务器上的数据库里，通过电子邮件发送信息，或者执行很多其他的功能。

有很多语言都可以用于编写表单处理脚本。对于刚起步的新手，PHP 是个不错的选择，因为用它处理一些常见任务很简单。而且，有很多关于 PHP 的书籍、在线教程、论坛帮你学习。关于 PHP 的内容超出了本书的范围，不过我在 www.htmlcssvqs.com/8ed/form-scripts 提供了两个基本的示例脚本（如图 16.3.1 所示）。

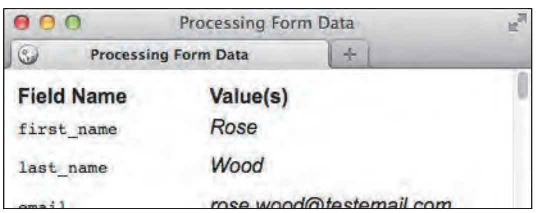


图 16.3.1　本书配套网站演示的脚本之一是 show-data.php。这里是它的一部分，提交表单后，它会将用户填写的所有字段的名和值显示出来。另一个脚本（email-data.php）会将表单数据发送至脚本中指定的电子邮件地址

除了 PHP，我们还可以选择其他语言，如 Django（一个使用 Python 的框架）、Ruby on Rails、ASP.NET、JSP（JavaServer Pages）等。

表单安全性

从服务器接收数据尤其需要注意安全性。不要对数据作任何假设，因为即便对表单建立了安全措施，坏人也可以创建他们自己的表单，调用你的脚本并发送无数的垃圾信息。他们还可能提交恶意文本，损坏服务器上的数据。保护表单是个前沿主题，我在 www.htmlcssvqs.com/8ed/form-security 提供了一些链接。

表单验证

表单验证指的是提交表单时，对用户输入的每个字段的内容进行检查，看是否符合预期的格式（例如，对于电子邮件字段，检查输入是否为正确的电子邮件地址格式）。先前提到，有的表单元素有内置的验证功能。有的网站使用 JavaScript 进行验证（网

上有相关的教程和代码）。这些都不能完全替代服务器端验证，因为旧的浏览器或禁用 JavaScript 的浏览器不会执行客户端验证。进行服务器端验证也是出于安全性考虑（参见前面的补充材料“表单安全性”）。总之，必须考虑将服务器端的验证加入表单处理脚本。

服务器端与客户端

PHP 是一种**服务器端**（server-side）语言，这意味着它运行于为网页服务的计算机（称为**服务器**），而不是查看网页的访问者的计算机。脚本必须上传到服务器上才能发挥作用，这通常跟你托管网页、图像等用的是同一个服务器（相关内容参见 21.2 节）。此外，服务器必须已经安装 PHP 才能对脚本进行解释。大多数 Web 主机都为你安装了 PHP，因此很容易找到支持 PHP 的服务器。专业网站的很多功能（如存储数据、发送电子邮件等）都需要服务器端语言。

客户端（client-side）语言（如 HTML 和 CSS）是在浏览器中运行的。JavaScript 是另一种客户端语言（也可以用在服务器端）。它们可以在完全不与服务器交互的情况下执行很多任务。例如，可以使用 JavaScript 在提交表单前检查是否所有数据都已填写，以及其他不需要服务器（或在涉及服务器之前）的任务。

16.4　对表单元素进行组织

如果表单上有很多信息需要填写，可以使用 `fieldset` 元素将相关的元素组合在一起，使表单更容易理解。表单越容易让访问者理解，访问者就越有可能正确地填写表单。还可以使用 `legend` 元素为每个 `fieldset` 提供一个标题（caption），用于描述每个组的目的，有时这些描述还可以使用 `h1` ~ `h6` 标题，如图 16.4.1 所示。对于一组单选按钮（参见 16.9 节），`legend` 元素尤其重要，因为通常如果不配合使用 `legend`，单选按钮（参见 16.9 节）就没有明显的上下文。

即便不添加任何 CSS，浏览器也会让哪些控件属于哪个 `fieldset` 显得相当清晰（如图 16.9.2 所示）。当然，你可以自己为 `fieldset` 和 `legend` 添加样式，从而让表单更吸引人，更便于使用（如图 16.9.3 和图 16.9.4 所示）。

对表单元素进行组织的步骤

(1) 在 `form` 开始标签的下面、任何希望包含在第一个组的表单元素的上面，输入 `<fieldset>`。

(2) 如果需要，输入 `<legend>`。（如果需要包含 `legend`，它必须是 `fieldset` 里的第一个元素。）

(3) 输入标签的文本。

(4) 输入 `</legend>` 以完成标签。

(5) 如果不使用 `legend`，则创建一个标题从而可以识别属于该 `fieldset` 的一组控件。（参见补充材料“legend 元素、标题、屏幕阅读器和样式”。）

(6) 创建属于第一组的表单元素，更多信息参见从 16.5 节开始讲解的知识。

(7) 输入 `</fieldset>` 结束第一组表单元素。

(8) 为每一组表单元素重复第 (1) 步至第 (7) 步。

```
...
<h1>Create a New Account</h1>
<form method="post" action="show-data.php">
    <fieldset>
       <h2 class="hdr-account">Account</h2>
       ... Account字段 ...
    </fieldset>

    <fieldset>
       <h2 class="hdr-address">Address</h2>
       ... Address字段 ...
    </fieldset>

    <fieldset>
       <h2 class="hdr-public-profile">Public Profile</h2>
       ... Public Profile字段 ...

       <div class="row">
          <fieldset class="radios">
             <legend>Gender:</legend>
             <input type="radio" id="gender-male" name="gender" value="male" />
             <label for="gender-male">Male</label>

             <input type="radio" id="gender-female" name="gender" value="female" />
             <label for="gender-female">Female</label>
          </fieldset>
       </div>
    </fieldset>

    <fieldset>
       <h2 class="hdr-emails">Emails</h2>
       ... Emails字段 ...
    </fieldset>

    <input type="submit" value="Create Account" class="btn" />
</form>
...
```


图 16.4.1　我对四个表单部分分别使用了 fieldset，并将 Public Profile（公开资料）部分的 Gender（性别）单选按钮使用一个 fieldset 包围起来。注意，我为被嵌套的 fieldset 添加了 radios 类，方便为其添加特定的样式，同时，还在其中添加了一个 legend 元素，用于描述单选按钮（为确保尽量简单，这段代码中移除了一些 div。这段被嵌套的 fieldset 的完整代码见 16.9 节）

提 示　使用 fieldset 元素对表单进行组织是可选的，使用 legend 也是可选的（使用 legend 则必须要有 fieldset）。不过我强烈推荐使用 fieldset 和 legend 对相关的单选按钮进行分组。

提 示　CSS 对 legend 元素的样式修改能力有限，尤其是关于定位的样式。如果遇到麻烦，可以上网搜索相关的解决办法。



图 16.4.2　没有添加任何 CSS 的情况下，你可以看到，浏览器默认为每个 fieldset 添加了一个边框，Public Profile 里面嵌套的 Gender 也有边框

```
fieldset {
    background-color: #f1f1f1;
    border: none;
    border-radius: 2px;
    margin-bottom: 12px;
    overflow: hidden;
    padding: 0 .625em;
}

.radios { /* nested fieldset */
    background-color: transparent;
    position: relative;
    margin-bottom: 0;
}

h2 {
    background-color: #dedede;
    border-bottom: 1px solid #d4d4d4;
    border-top: 1px solid #d4d4d4;
    border-radius: 5px;
    box-shadow: 3px 3px 3px #ccc;
    color: #fff;
    font-size: 1.1em;
    margin: 12px;
    padding: 0.3em 1em;
    text-shadow: #9FBEB9 1px 1px 1px;
    text-transform: uppercase;
}

.hdr-account { background-color: #0b5586; }
.hdr-address { background-color: #4494c9; }
.hdr-public-profile { background-color:
→ #377d87; }
.hdr-emails { background-color: #717f88; }
```

图 16.4.3　我为所有的 fieldset 元素添加了外边距、背景颜色和内边距，并为每个标题添加了特殊的背景颜色

legend 元素、标题、屏幕阅读器和样式

legend 元素可以提高表单的可访问性。对于每个表单字段，屏幕阅读器都会将与之关联的 legend 文本念出来，从而让访问者了解字段的上下文。这种行为在不同的屏幕阅读器和浏览器上并不完全一样，不同的模式下也不一样（屏幕阅读器的用户有多种听读和导航的模式）。

很多情况下，legend 文本会如你所预期的那样念出来。有时，文本不会念出来；有时，屏幕阅读器又会走向另一个极端，针对每个控件都将 legend 文本念一遍。此外，有的浏览器会限制可以对 legend 添加的样式，或者让添加样式变得更难。

考虑到以上这些问题，使用 h1 ~ h6 标题代替 legend 来识别一些（并非所有）fieldset，就像我之前所做的那样，参见图 16.4.1，就是一个明智的选择。JAWS（用户数量最多的屏幕阅读器）会将标题与相关联的表单字段连在一起阅读，因此可访问性不会受到影响。而且，屏幕阅读器允许用户通过标题在页面内切换内容。此外，还可以很轻松地对标题添加样式，这也是一个额外的好处。

并非所有的表单都是相似的，因此你所使用的方法可以有一些差异。无论如何，对于单选按钮，最好总是使用 fieldset 和 legend。

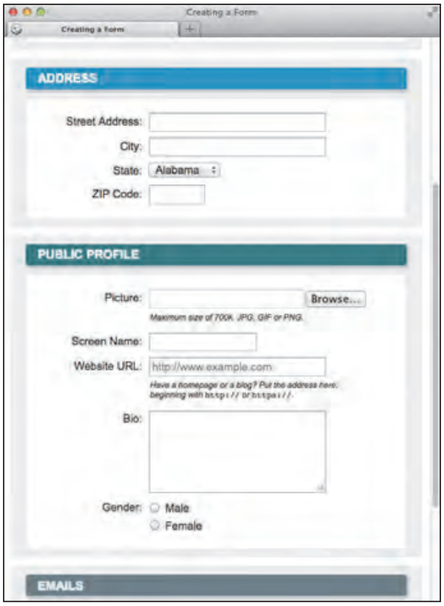


图 16.4.4　新添加的 CSS 将不同分组的表单字段清晰地隔开了

16.5　创建文本框

文本框可以包含一行无格式的文本，它可以是访问者想输入的任何内容，通常用于姓名、地址等信息。

每个文本框都是通过带有 type="text" 的 input 标签表现出来的。除了 type 之外，还有一些其他可用的属性，其中最重要的就是 name（参见图 16.5.1）。服务器端的脚本使用你指定的 name 获取访问者在文本框中输入的值或预设的值（即 value 属性值）。事实上，name 和 value 对其他的表单字段类型来说，也是很重要的，你将在本章余下的部分看到这一点。

```
...
<form method="post" action="show-data.php">
<fieldset>
    <h2 class="account">Account</h2>
    <div class="fields">
        <p class="row">
            <label for="first-name">First Name:
            → </label>
            <input type="text" id="first-
            name" name="first_name"
            → class="field-large" required
            → aria-required="true" />
        </p>
        <p class="row">
            <label for="last-name">Last Name:
            → </label>
            <input type="text" id="last-
            → name" name="last_name"
            → class="field-large" />
        </p>
        <p class="row">
            <label for="email">Email:</label>
            <input type="email" id="email"
            → name="email" placeholder=
            → "yourname@example.com"
            → class="field-large" />
        </p>
        ... 更多字段 ...
    </div>
...
```

图 16.5.1　必须为每个文本框设置 name 属性，只有在希望为文本框添加默认值的情况下才需要设置 value 属性。第三个文本框使用了 placeholder 属性（参见图 16.5.2），第一个则使用了 required 属性（参见图 16.5.3）。注意，我还设置了 aria-required="true"（参见提示）。这个例子还演示了 name 可以与 for 和 id 不一样，也可以与它们一样（参见 16.6 节的补充材料）

创建文本框的步骤

(1) 如果需要，输入用于让访问者识别文本框的标签，例如 `<label for="idlabel">Last Name:</label>`，其中 *idlabel* 匹配第 (4) 步中的标签，关于这一点，下一节会给出更多解释。

(2) 输入 `<input type="text"`。

(3) 输入 name="*dataname*"，这里的 *dataname* 是用于让服务器（和脚本）识别输入数据的文本。

(4) 如果在第 (1) 步中创建了 label，输入 id="*idlabel*"，这里的 *idlabel* 跟第 (1) 步中 for 属性的文本一样。这跟具有显式文本框的 label 元素关联。很多程序员将 id 和 name 设为同一个值，尽管这并不是必需的（图 16.5.1 展示了这两种方法）。

(5) 如果需要，输入 value="*default*"，这里的 *default* 是这个字段中最初显示的数据，如果访问者没有输入别的内容的话，这一数据将被发送到服务器。

(6) 如果需要，输入 placeholder="*hinttext*"，这里的 *hinttext* 是这个字段中最初显示的数据，用于指导用户的输入，如图 16.5.2 所示。当 input 元素获得焦点时，这些文本将会消失，让用户输入内容。

(7) 如果需要，输入 required="required"，表示仅在这个字段有值的情况下才能提交表单，如图 16.5.3 所示。

(8) 如果需要，输入 autofocus 或 autofocus="autofocus"（在 HTML5 中这两种方法均可），如图 16.5.4 所示。如果这是第一个拥有此属性的表单控件，input 元素在页面加载时会默认获得焦点。

(9) 如果需要，通过输入 size="*n*" 定义文本框的大小，这里的 *n* 是需要设置的文本框宽度，以字符为单位。也可以使用 CSS 设置输入框的宽度，参见图 16.5.5。

(10) 如果需要，输入 maxlength="*n*"，这里的 *n* 是该文本框允许输入的最大字符数。

(11) 最后，输入 > 或 />，结束文本框（参见最后一条提示）。

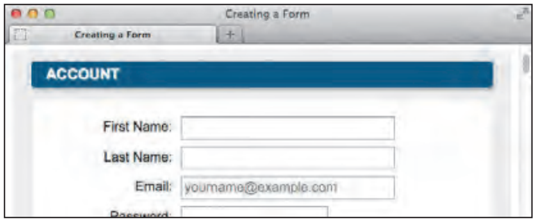


图 16.5.2　占位符是为用户填写表单提供提示和额外指导的好方法。placeholder 属性中的文本会以浅灰色出现在文本框中。当用户开始在字段中输入文本时，浅灰色文本就会消失。（一种例外情况是：当字段获得焦点，而非用户开始输入的时候，IE10 会隐藏占位符文本。）如果用户在没有输入任何信息的情况下将焦点移开，浅灰色文本又会再次出现。这是 HTML5 的另一项新特性，旧浏览器会忽略它

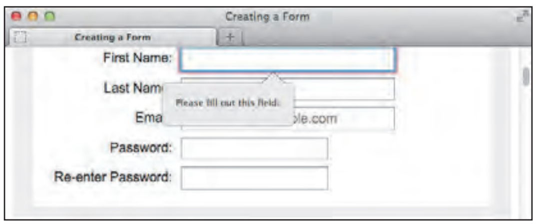


图 16.5.3　如果用户在提交表单时有一个带 required 属性的字段没有填写，浏览器会显示一个类似的提示。这个消息的样式在不同的浏览器中会有差异。这个特性是 HTML5 专有的，因此旧的浏览器会忽略 required（不过表单仍可正常工作）。在 16.3 节中提到，针对旧的浏览器，可以使用 JavaScript 对表单进行验证，此外，无论用户使用哪种浏览器，都应该在服务器端验证表单

```
<input type="text" id="first-name"
→ name="first_name" class="field-large"
→ required aria-required="true" autofocus />
```

图 16.5.4　页面加载时，最好有一个字段自动获取焦点，这样用户就可以立即开始输入。要实现这一功能，可以使用 autofocus 属性

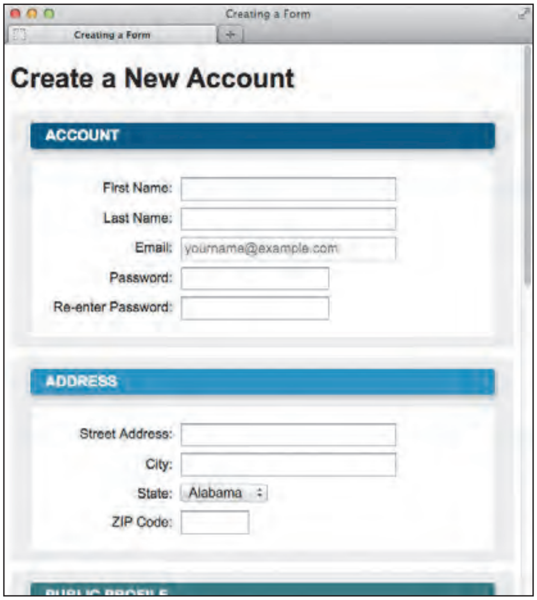


图 16.5.5　针对不同的字段类型，可以使用不同尺寸的文本框。在我们的例子中，我们使用 CSS 通过类设置宽度

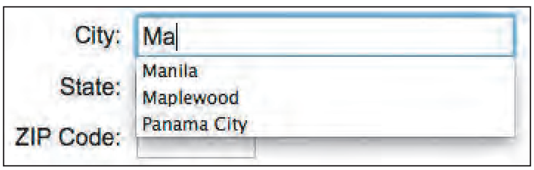


图 16.5.6　输入 Ma 后，Firefox 根据先前的表单输入给出了一些包含同样字母的建议文本。我可以从提示文本列表中进行选择，也可以继续输入。如果整个表单或者这个特定的 input 设置了 autocomplete="off"，就不会显示这些提示，浏览器也不会保存用户所输入的内容

分隔表单元素

对各个表单元素进行分隔有很多方法。在这些例子中，我们大多数情况下使用 p 元素，还有一些情况下使用 div 元素（譬如嵌套 p 的情况），参见图 16.5.1。所有的示例中都使用了 class="row" 作为定义样式的钩子。此外，有的人使用 ol 或 ul 来组织表单的元素。通常，列表对屏幕阅读器用户来说很有帮助。不过有的屏幕阅读器在朗读放在列表中的表单元素时会有一些问题，并让用户感到疑惑。因此，我选择 p 和 div。

提示　尽管 label 是可选的，但我强烈推荐使用它们。它们对提升表单的可访问性和可用性有很重要的作用。

提示　如果访问者跳过了一个字段，而且你没有为该字段设置默认值，那么当用户提交表单时，发送到服务器的 name 属性就是未定义的空值。

提示　不要将 placeholder 属性（图 16.5.1 和图 16.5.2 中都有例子）同 value 属性弄混。它们都会让文本框默认出现一些文本，但 placeholder 文本会自动消失，且不会被发送到服务器，而 value 在输入框获得焦点时不会消失，且这些内容会被发送到服务器。由此看来，如果一个文本框同时具有 placeholder 和非空 value，后者会显示在文本框中。

提示　默认情况下，大多数浏览器会保存用户输入的文本，这样就可以在日后节省用户输入的时间（参见图 16.5.6）。可以通过对 input 添加 autocomplete="off" 来关闭这一特性。这对于要求输入敏感信息（如信用卡号）的字段来说是很有用的。如果将它用于 form 元素（例如，<form method="post" action="process.php" autocomplete="off">），那么其中的每个字段都会这样。

提示　我们所说的“获取焦点”或“拥有焦点”，指的是光标位于文本框内，访问者可以随时输入值的状态。

> **提示**　文本框大小的默认值为 20。不过，访问者可以输入更多的字符，直到达到 maxlength 属性规定的限制。如果需要更大的可以容纳好几行文本的字段，最好使用 textarea 元素（参见 16.11 节）。

> **提示**　就像其他自关闭的元素（如 img）一样，使用 > 或 /> 结尾在 HTML5 中都是有效的。无论你选择哪种方式，一定要保持一致。

16.6　为表单组件添加说明标签

标签（label）是描述表单字段用途的文本。例如，在访问者应该输入其姓名中名的文本字段之前，可能有“First Name:”的字样。可以使用 label 元素（有点小意外吧）标记这些文字说明标签。你或许已经注意到前面的示例中已经使用过 label 元素了。

label 元素有一个特殊的属性：for。如果 for 的值与一个表单字段的 id 的值相同，该 label 就与该字段显式地关联起来了（如图 16.6.1 所示）。这对提升表单的可用性和可访问性都有帮助。例如，如果访问者与标签有交互（如使用鼠标点击了标签），与之对应的表单字段就会获得焦点（如图 16.6.2 所示）。这种关联还可以让屏幕阅读器将文本标签与相应的字段一起念出来。设想，这对不了解表单字段含义的视障用户来说是多么重要。出于这些原因，我强烈建议你在 label 元素中包含 for 属性。

```
...

<fieldset>
    <h2 class="account">Account</h2>
    <div class="fields">
        <p class="row">
            <label for="first-name">
            → First Name:</label>
            <input type="text"
            → id="first-name"
            → name="first_name"
            → class="field-large" />
        </p>
        <p class="row">
            <label for="last-name">
            → Last Name:</label>
            <input type="text"
            → id="last-name"
            → name="last_name"
            → class="field-large" />
        </p>
        ... 更多字段 ...
    </div>
</fieldset>

...
```

图 16.6.1　每个 label 的 for 属性与该标签对应的表单字段的 id 是匹配的，这样可以将该标签与该字段显式关联起来

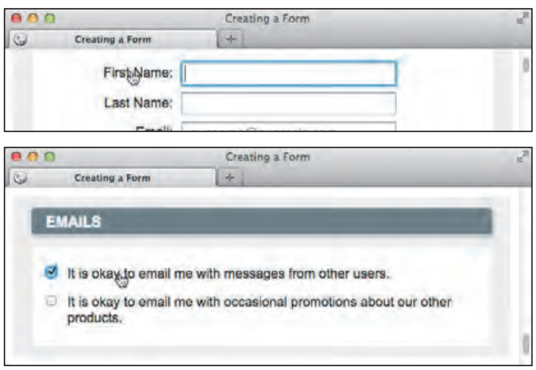


图 16.6.2　如果访问者与文本框的标签有交互，光标就会位于该文本框内，方便用户输入。同时，复选框和单选按钮的标签则让用户可以通过点击标签来修改状态（就像点击表单控件一样）。这里，选择了一个复选框

为表单组件添加正式标签

(1) 输入 <label。

(2) 如果需要，输入 for="*idlabel*">，这里的 *idlabel* 是对应表单元素的 id 属性值。

(3) 输入标签的内容。

(4) 输入 </label>。

(5) 如果创建的是表单元素，确保包括 id 属性，对应于第 (2) 步中指定的 for 属性。

提示 可以使用 CSS 对标签添加样式。图 16.6.3 的示例让每个标签都与它旁边的字段对齐（参见图 16.6.2）。

提示 for、id 和 name 属性都可以拥有任意值，只要没有空格。相关信息参见本页补充材料中的介绍。

提示 还可以将一个表单字段放在一个包含标签文本的 label 内，例如，<label>First Name: <input type="text" name="first_name" /></label>。（注意在这种情况下，就不需要使用 for 和 id 了。）不过，将标签与字段分开是更常见的做法（参见图 16.6.1），原因之一是这样更容易添加样式。

提示 有时，placeholder 会被误作 label 的替代属性。一定要确保 placeholder 只能用作一种提示。

```
/* 表单字段前面的标签 */
label {
    cursor: pointer;
    display: inline-block;
    padding: 3px 6px;
    text-align: right;
    width: 150px;
    vertical-align: top;
}

/* 复选框后面的标签 */
.checkboxes label {
    text-align: left;
    width: 475px;
}
```

图 16.6.3 对标签设置样式，是让表单变得更方便使用和更好看的重要方面。如果为 label 指定了 cursor: pointer;，当访问者指向标签时，就会显示为手形，而不是默认的箭头。显示为手形就能提示用户这是一个可以操作的元素。使用 vertical-align: top; 则会让标签与相关的表单字段对齐

16

id、for 和 name 属性的命名习惯

正如先前提到的，让 for、id 和 name 属性值都一样是一种并非必需但很常见的做法。（单选按钮和复选框是例外，因为对它们来说，有一组 input 会使用同一个 name，而 id 要求对每个 input 来说都是唯一的。）本章的例子中使用的值都是一个单词，例如 for="email"、id="email" 和 name="email"。

对于包含多个单词的值，我们在 for 和 id 中使用连字符（-）分隔各个单词，在 name 中使用下划线（_），参见图 16.6.1。例如，for="first-name"、id="first-name" 和 name="first_name"。对 name 使用不同的形式，表现出该属性的值可以与另外两个不同，实际上对需要传入服务器端脚本的 name 值使用下划线分隔多个单词也是一种常见的做法。

无论使用哪种方法，一定要保证 for 和 id 的值是一样的。

16.7　创建密码框

密码框与文本框的唯一区别是，密码框中输入的文本会使用圆点或星号进行隐藏，如图 16.7.1 和图 16.7.2 所示。

```
...
<p class="row">
    <label for="password">Password:</label>
    <input type="password" id="password"
    → name="password" />
</p>
<p class="row">
    <label for="password2">Re-enter
    → Password:</label>
    <input type="password" id="password2"
    → name="password2" />
</p>
...
```

图 16.7.1　使用 type="password" 创建密码框，而不要用 type="text"

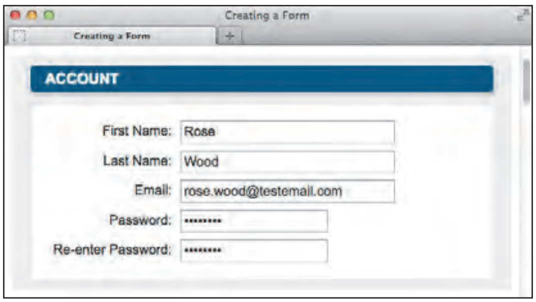


图 16.7.2　当访问者在表单中输入密码时，密码用圆点或星号隐藏起来了。但提交表单后访问者输入的真实值会被发送到服务器。信息在发送过程中没有加密

创建密码框的步骤

(1) 输入用于让访问者识别密码框的标签，跟 16.6 节中的一样。

(2) 输入 <input type="password"。

(3) 输入 id="*idlabel*"，这里的 *idlabel* 跟第 (1) 步中 label 的 for 属性值一样。

(4) 输入 name="*dataname*"，这里的 *dataname* 是用于让服务器识别输入数据的文本。

(5) 通过输入 size="*n*" 定义密码框的大小，这里的 *n* 是需要设置的密码框宽度，以字符为单位。

(6) 如果需要，输入 maxlength="*n*"，这里的 *n* 是该密码框允许输入的最大字符数。

(7) 如果需要，输入 required 或者 required="required"（参见 16.5 节）。

(8) 如果需要，输入 autofocus 或者 autofocus="autofocus"（参见 16.5 节）。

(9) 输入 > 或 />，结束密码框。

提示　即便密码框中没有输入任何内容，*name* 属性仍将被发送到服务器（使用未定义的 *value*）。

提示　密码框提供的唯一保护措施就是防止其他人看到用户输入的密码。如果要真正地保护密码，可以使用安全服务器（https://）。

16.8　创建电子邮件框、搜索框、电话框和 URL 框

电子邮件、电话和 URL 这几种输入类型是 HTML5 中新增的，如图 16.8.1 所示。它们看起来同文本框很相似，但却有一些小而有用的特性，用于帮助验证和输入内容，如图 16.8.2 ~ 图 16.8.4 所示。过去，我们必须依赖 JavaScript 在浏览器中创建这些功能。

搜索框也是 HTML5 中新增的输入类型（如图 16.8.5 所示）。搜索框跟文本框很像，只是很多浏览器将搜索框显示得像操作系统中默认的搜索框，参见图 16.8.6。

```
...
<p class="row">
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" class="field-large" />
</p>
<div class="row">
    <label for="website">Website URL:</label>
    <input type="url" id="website" name="website" class="field-large"
    ➝ placeholder="http://www.example.com" />
    <p class="instructions">Have a site or a blog? Put the address here, beginning with
    ➝ <kbd>http://</kbd> or <kbd>https://</kbd>.</p>
</div>
<p class="row">
    <label for="phone">Phone:</label>
    <input type="tel" id="phone" name="phone" class="field-large" placeholder="xxx-xxx-xxxx"
    ➝ pattern="\d{3}-\d{3}-\d{4}" />
</p>
...
```

图 16.8.1　合适的 type 属性值分别指定电子邮件框、URL 框和电话框。pattern 属性用于定制的验证规则。它使用正则表达式对用户在框中输入的内容进行限制。对于这个例子中的电话输入框，pattern 属性要求“只接受以下格式的输入：*xxx-xxx-xxxx*，其中 *x* 均为数字“（许多电话号码都是这样的）。不必对那些不同寻常的正则表达式语法感到担心，可以在 http://html5pattern.com 找到一些常用的正则表达式，并将它们复制粘贴到自己的 pattern 属性中

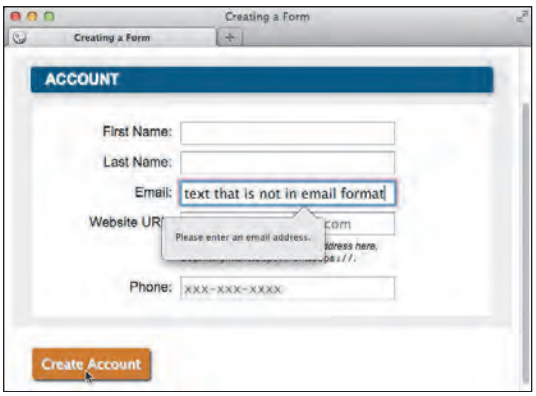


图 16.8.2　当访问者在现代浏览器中提交表单时，浏览器会对 Email 字段的输入文本进行检查，确保其符合电子邮件地址的格式。如果格式不合法，会显示如图所示的错误信息，且鼠标重置到该字段，以方便访问者修改文本



图 16.8.3　当访问者在现代浏览器中提交表单时，浏览器会对 Website URL 字段的输入文本进行检查，确保其符合 URL 格式。注意，www.wikipedia.org 并不是有效的 URL，因为 URL 必须以 http:// 或 https:// 开头。这里最好使用占位符提示访问者。另外，我还在该字段下面的解释文本中指出了合法的格式。从图 16.8.4 可以看得很清楚

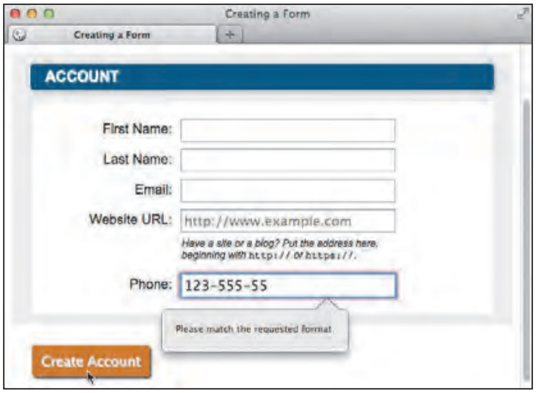


图 16.8.4　当访问者在现代浏览器中提交表单时，浏览器会对 Phone 字段的输入文本进行检查，确保其符合 pattern 属性中指定的电话号码格式。同时，这种字段对于使用 iOS（iPhone 和 iPad）上的 Safari 的用户来说也很方便，因为它会启动数字键盘而非常见的标准键盘

```
<form method="get" action="search-results.php"
→ role="search">
    <label for="search">Search:</label>
    <input type="search" id="search"
    → name="search" size="30"
    → placeholder="e.g., a book or
    → magazine" />
    <input type="submit" value="Find It!" />
</form>
```

图 16.8.5　搜索框是应用 placeholder 的天然场合。同时，注意这里的 form 用的是 method="get" 而不是 method="post"。这是搜索字段的常规做法（无论是 type="search" 还是 type="text"）。关于 role="search"，参见提示

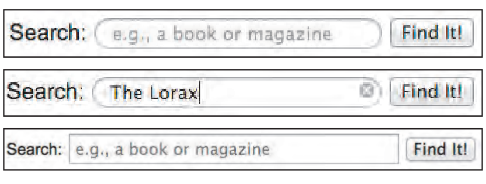


图 16.8.6　OS X 上的 Chrome（如上面的两幅图）、Safari 以及 iOS 上的 Mobile Safari 会让搜索框显示为圆角边框。当用户开始输入，字段右侧会出现一个 x 按钮，用于清除输入的内容。在其他浏览器中，它显示为常规文本框的样子，如最下面一张图（更多关于样式的讨论，参见提示）

创建电子邮件框、搜索框、电话框和 URL 框的步骤

(1) 输入用于让访问者识别输入框的标签，参见 16.6 节。

(2) 对于电子邮件框，输入 <input type="email"；对于搜索框，输入 <input type="search"；对于电话框，输入 <input type="tel"；对于 URL 框，输入 <input type="url"。

(3) 输入 id="*idlabel*"，这里的 *idlabel* 跟第 (1) 步中 label 的 for 属性相同。

(4) 输入 name="*dataname*"，这里的 *dataname* 是用于让服务器识别输入数据的文本。

(5) 如果需要，输入 value="*default*"，这里的 *default* 是最初显示在字段里的数据，也是访问者什么都不输入时将要发送给服务器的数据。

(6) 如果需要，输入 placeholder="*hinttext*"，这里的 *hinttext* 是最初显示在字段里作为对用户输入内容的提示的数据（参见 16.5 节）。

(7) 如果需要，输入 required 或者 required="required"（参见 16.5 节）。

(8) 如果需要，输入 autofocus 或者 autofocus="autofocus"（参见 16.5 节）。

(9) 如果需要，通过输入 size="*n*" 定义框的大小，这里的 *n* 是需要设置的框的宽度，以字符为单位。也可以使用 CSS 设置输入框的宽度（参见 16.5 节）。

(10) 如果需要，输入 maxlength="*n*"，这里的 *n* 是该框允许输入的最大字符数。

(11) 最后，输入 > 或者 />，结束输入框（两种方法在 HTML5 中均可）。

提示 截至本书写作之际，Chrome、Firefox、IE10 和 Opera 10+ 都提供了对电子邮件框和 URL 框的验证功能。注意这些功能只是为网站创立者和访问者提供了一些便利，但服务器端的验证还是很有必要的（参见 16.3 节）。即便你使用像之前提到过的 H5F（https://github.com/ryanseddon/H5F）这样的 JavaScript 来为旧浏览器提供 HTML5 功能，也必须建立服务器端验证。

提示 空的电子邮件框、电话框或 URL 框也会通过验证，除非有 required 属性。不过，如果对电话框添加了 required，但没有为 pattern 属性提供内容，浏览器仍会要求用户填写内容，但任何文本（包括数字、字母、字符等）都会通过验证。

提示 电子邮件框也可以使用 multiple 属性，这样就可以输入多个电子邮件地址（以逗号分隔不同的地址）。

提示 浏览器不会检查输入的电子邮件地址或 URL 是否真实存在，它只检查格式是否正确。

提示 这些输入类型也支持 autocomplete 属性，更多信息参见 16.5 节的提示。

提示 默认情况下，为 Chrome、Safari 和 Mobile Safari 等浏览器中的搜索框设置样式是受到限制的（参见图 16.8.6）。如果要消除这一约束，重新获得 CSS 的控制权，可以使用专有的 -webkit-appearance: none; 声明，例如 input[type="search"] { -webkitappearance: none; }。更多信息（包括对 Firefox 的支持）参见 http://css-tricks.com/almanac/properties/a/appearance/。不过要注意，appearance 属性并不是官方的 CSS，因此不同浏览器的行为有可能不一样。

提示 如果仔细查看图 16.8.5 中的代码，你会发现我为 form 添加了 ARIA 地标角色，role="search"。这会让屏幕阅读器指出网页上有搜索区域，从而提高页面的可访问性。如果你的表单有多个控件而非只有搜索，就将与搜索相关的控件放在一个 fieldset 或 div 中并为其添加 role="search"，而不是将 role="search" 添加到 form 元素本身。

提示 WebKit 浏览器支持两个非官方 HTML5 属性：autosave 和 results，它们为搜索框提供了额外的行为和视觉效果。更多信息参见 www.wufoo.com/html5/types/5-search.html。

提示 如果包含了 pattern 属性，一定要明确告诉访问者需要遵循的模式。稍不留意，访问者就有可能放弃提交表单。

提示 正则表达式超出了本书的范围，不过，网上有很多在线资源（搜索 regex tutorial）。http://html5pattern.com 上有很多有用的模式[①]。

① 模式指的是用于描述一系列用来匹配规则的字符串（表达式）。——译者注

16.9　创建单选按钮

还记得老式汽车收音机上那些大大的黑色塑料按钮吗？按下其中的一个可以收听 WFCR；按下另一个可以收听 WRNX。不过，不可以同时按下两个按钮。单选按钮也遵循同样的工作方式。对 input 元素设置 type="radio" 即可创建单选按钮，参见图 16.9.1 和图 16.9.2。

```
...

<fieldset class="radios">
    <legend>Gender:</legend>

    <p class="row">
       <input type="radio"
       ➝ id="gender-male" name="gender"
       ➝ value="male" />
       <label for="gender-male">Male</label>
    </p>
    <p class="row">
       <input type="radio"
       ➝ id="gender-female" name="gender"
       ➝ value="female" />
       <label for="gender-female">Female
       ➝ </label>
    </p>
</fieldset>

...
```

图 16.9.1　同一组单选按钮的 name 属性值必须相同，这样在同一时间只有其中一个能被选中。value 属性也很重要，因为对于单选按钮访问者无法输入值

```
.radios {
    background-color: transparent;
    position: relative;
    margin-bottom: 0;
}

.radios .row {
    margin: 0 0 0 150px;
}

.radios legend {
    left: 0;
    padding: 0 6px;
    position: absolute;
    text-align: right;
    top: 2px;
    width: 148px;
}

.radios label {
    padding-left: 2px;
    margin-right: 5px;
    vertical-align: middle;
    width: auto;
}
```

图 16.9.2　这里让 legend 在 .radios div 中绝对定位，.radios div 已设为 position: relative（对 legend 来说，position 比 margin 更有效）。为 .row 类添加一个大的 margin-left 值，让它们位于 Gender 的 legend 的右侧。对 label 元素设置的 vertical-align: middle; 则让标签文本与单选按钮在竖直方向上对齐

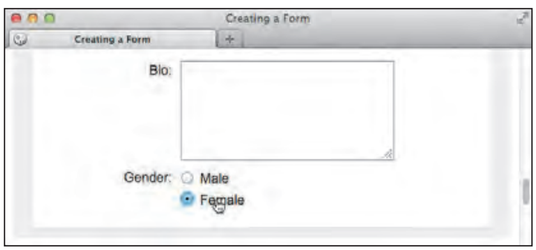


图 16.9.3　由于标签（Male 和 Female）是 label 元素，与标签交互则会选中它们对应的单选按钮

创建单选按钮的步骤

(1) 如果需要，输入单选按钮的介绍文本。例如，可以使用“选择下列选项中的一个”。

(2) 输入 <input type="radio"。

(3) 输入 name="*radioset*"，这里的 *radioset* 用于识别发送至服务器的数据，同时用于将多个单选按钮联系在一起，确保同一组中最多只有一个被选中。

(4) 输入 id="*idlabel*"，这里的 *idlabel* 对应你在第 (8) 步中创建的 for 属性值。跟 name 值不同（同一组单选按钮的 name 值都是

相同的），同一页面中每个元素的 id 必须是唯一的。

(5) 输入 value="*data*"，这里的 *data* 是该单选按钮被选中（无论是被默认选中还是被访问者选中）时要发送给服务器的文本（如图 16.9.3 所示）。

(6) 如果需要，输入 checked 或者 checked="checked" 让该单选按钮在页面打开时默认处于激活状态（两种方法在 HTML5 中均可）。在一组单选按钮中，只能对一个按钮添加这一属性。

(7) 输入 > 或者 />。

(8) 输入<label for="*idlabel*">*radio label*</label>，其中，*idlabel* 与第 (4) 步中单选按钮的 id 值相同，*radio label* 则用于让访问者识别该单选按钮。*radio label* 的值通常与 value 的值相同，但这并不是必需的。

(9) 对同一组内的所有单选按钮，重复第 (2) 步至第 (8) 步。

提 示　我推荐使用 fieldset 嵌套每组单选按钮，并用 legend 进行描述（如图 16.9.1 所示）。更多细节参见 16.4 节。

16.10　创建复选框

在一组单选按钮中，只允许选择一个答案；但在一组复选按钮中，访问者可以选择任意数量的答案。同单选按钮一样，复选框也与 name 属性的值联系在一起，参见图 16.10.1。

```
<div class="fields checkboxes">
    <p class="row">
        <input type="checkbox" id="email-ok-msg-from-users" name="email_signup[]"
        ➝ value="user-emails" />
        <label for="email-ok-msg-from-users">It is okay to email me with messages from other users.
        ➝ </label>
    </p>
    <p class="row">
        <input type="checkbox" id="email-ok-occasional-updates" name="email_signup[]"
        ➝ value="occasional-updates" />
        <label for="email-ok-occasional-updates">It is okay to email me with occasional promotions
        ➝ about our other products.</label>
    </p>
</div>
```

图 16.10.1　注意，标签文本（未突出显示）不需要与 value 属性一致。这是因为标签文本用于在浏览器中向访问者标识复选框，而 value 则是发送到服务器端脚本的数据的一部分。空的方括号是为 PHP 脚本的 name 准备的（参见提示）。我创建了一个 .checkboxes 类，以限制为复选框添加样式的 label，参见图 16.10.2

```
.checkboxes label {
    text-align: left;
    width: 475px;
}
```

图 16.10.2　对于复选框，通常需要为标签设置不同的样式，因为这些标签位于输入元素之后，参见图 16.10.3

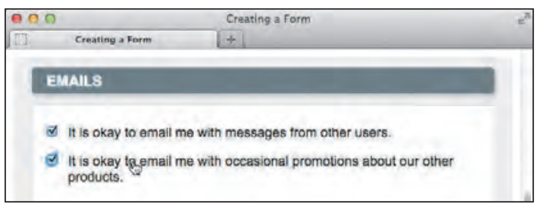


图 16.10.3　访问者可以根据需要选择任意数量的框，每个框对应的值及复选框组的名称都会被发送到脚本

创建复选框的步骤

(1) 如果需要，输入复选框的介绍文本。例如，可以使用“选择下列选项中的一个或多个”。

(2) 输入 `<input type="checkbox"`。

(3) 输入 name="*boxset*"，这里的 *boxset* 用于识别发送至服务器的数据，同时用于将多个复选框联系在一起（对于所有复选框使用同一个 name 值）。

(4) 输入 id="*idlabel*"，这里的 *idlabel* 对应于第 (8) 步中 label 的 for 属性值。

(5) 输入 value="*data*"，这里的 *data* 是该复选框被选中（无论是被访问者选中还是被建站者选中，如图 16.10.1 所示）时要发送给服务器的文本。

(6) 输入 checked 或 checked="checked" 让该复选框在页面打开时默认处于选中状态（在 HTML5 中两种写法均可），建站者或访问者可能会勾选默认的选项。

(7) 输入 > 或者 /> 以结束复选框。

(8) 输入 <label for="*idlabel*"> *checkbox label*</label>，其中，*idlabel* 与第 (4) 步复选框元素中的 *id* 值相同，*checkbox label* 则用于让访问者识别该复选框。

(9) 对同一组内的所有复选框，重复第 (2) 步至第 (8) 步。

提 示　如果使用 PHP 处理表单，在图 16.10.1 中使用 name="*boxset*[]"（这里的 *boxset* 用于标识发送给脚本的数据）就会自动地创建一个包含复选框值的数组（名为 $_POST['boxset']）。

16.11　创建文本区域

如果希望给访问者填写问题或评论的空间，可以使用文本区域。参见图 16.11.1。

```
<label for="bio">Bio:</label>
<textarea id="bio" name="bio" cols="40"
➝ rows="5" class="field-large"></
textarea>
```

图 16.11.1　rows 和 cols 属性分别控制文本区域的高度和宽度，除非使用 CSS 覆盖其中之一（参见图 16.11.2）或者全部。即便在 CSS 中设置了尺寸，rows 和 cols 还是有用的，因为用户的浏览器有可能会关闭 CSS

```
textarea {
    font: inherit;
    padding: 2px;
}

.field-large {
    width: 250px;
}
```

图 16.11.2　默认情况下不会继承 textarea、font 属性，因此必须显式地设置该属性。如果需要，可以为与其他文本框、URL 框等输入框具有一个类（这里为 .field-large）的文本区域设置宽度。由于没有设置高度，因此文本区域的高度（如图 16.11.3 所示）由 HTML 中的 rows 属性决定（参见图 16.11.1）

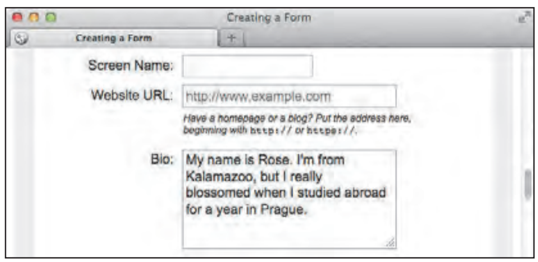


图 16.11.3　如果没有使用 maxlength 属性限制文本区域的最大字符数量，访问者可以输入多达 32 700 个字符。必要时，文本区域内会显示一个滚动条（这里没有显示）。访问者可以通过拖曳文本区域右下角的斜线改变文本区域的大小。如果设置了 textarea { resize: none; }，那么访问者就无法这样操作了

创建文本区域的步骤

(1) 如果需要，输入用于标识文本区域的解释性文本。

(2) 输入 `<textarea`。

(3) 输入 `id="`*`idlabel`*`"`，这里的 *`idlabel`* 跟第 (1) 步中 `label` 的 `for` 属性值相同。

(4) 输入 `name="`*`dataname`*`"`，这里的 `dataname` 是用于让服务器（和脚本）识别输入数据的文本。

(5) 如果需要，输入 `maxlength="`*`n`*`"`，这里的 *`n`* 是可以输入的最大字符数。

(6) 输入 `cols="`*`n`*`"`，这里的 *`n`* 是文本区域的宽度（以字符为单位）。

(7) 输入 `rows="`*`n`*`"`，这里的 *`n`* 是文本区域的高度（以行为单位）。

(8) 输入 `>`。

(9) 输入文本区域的默认文本（如果有的话），这些文本会显示在文本区域中。

(10) 输入 `</textarea>` 以完成文本区域。

提示 要为文本区域添加预设值，就在 `textarea` 开始标签和结束标签之间添加文本（这里没有 `value` 属性）。通常，可以包含 `placeholder` 属性定义用于占位的文本。

提示 `maxlength` 是 HTML5 中为文本区域新增的属性，因此它的行为在不同浏览器中可能不一样（参见 www.wufoo.com/html5/attributes/03-maxlength.html）。旧浏览器会直接忽略该属性。

提示 通过 CSS 可以更好地控制文本区域的尺寸。

16.12 创建选择框

选择框非常适合向访问者提供一组选项，从而允许他们从中选取，参见图 16.12.1。它们通常呈现为下拉菜单的样式，参见图 16.12.3。如果允许用户选择多个选项，选择框就会呈现为一个带滚动条的项目框。

```
<label for="state">State:</label>
<select id="state" name="state">
    <option value="AL">Alabama</option>
    <option value="AK">Alaska</option>
    ...
</select>
```

图 16.12.1 选择框由两种 HTML 元素构成：`select` 和 `option`。通常，在 `select` 元素里设置 `name` 属性，在每个 `option` 元素里设置 `value` 属性。我们可以为 `select`（如图 16.12.2 所示）和 `option` 元素添加样式，但有一定的限制

```
select {
    font-size: inherit;
}
```

图 16.12.2 CSS 规则要求菜单文本跟其父元素字号大小相同，否则默认情况下它看上去会小很多。可以使用 CSS 对 `width`、`color` 和其他的属性进行调整，不过，不同的浏览器呈现下拉菜单列表的方式略有差异

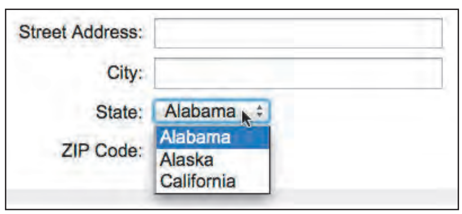


图 16.12.3 默认的选择是菜单中的第一个选项，或者是在 HTML 中指定了 `selected` 的选项（需要注意的一点是，除非设置了 `size` 属性，否则访问者就必须选择菜单中的某个选项）

1. 创建选择框的步骤

(1) 如果需要，输入描述菜单的文本，参见 16.6 节。

(2) 输入 <select。

(3) 输入 id="*idlabel*"，这里的 *idlabel* 跟第 (1) 步中 label 的 for 属性值一样。

(4) 输入 name="*dataname*"，这里的 *dataname* 用于在收集的数据发送至服务器时对数据进行识别。

(5) 如果需要，输入 size="*n*"，这里的 *n* 代表选择框的高度（以行为单位）。

(6) 如果需要，输入 multiple 或者 multiple="multiple"（两种方法在 HTML5 中均可），从而允许访问者选择一个以上的菜单选项（选择的时候需按住 Control 键或 Command 键）。

(7) 输入 >。

(8) 输入 <option。

(9) 输入 value="*optiondata*"，这里的 *optiondata* 是选项选中后要发送给服务器的数据（如果省略 value，你在第 (12) 步中输入的文本就是选项的值）。

(10) 如果需要，输入 selected 或者 selected="selected"（在 HTML5 中两种方式均可），指定该选项被默认选中。

(11) 输入 >。

(12) 输入希望出现在菜单中的选项名称。

(13) 输入 </option>。

(14) 对每个选项重复第 (8) 步至第 (13) 步。

(15) 输入 </select>。

如果有一个选项很多的特别大的菜单，可能需要对这些选项进行分组，分别放入不同的类别，参见图 16.12.4 和图 16.12.5。

2. 对选择框选项进行分组

(1) 根据“创建选择框的步骤”，创建所需的选择框。

(2) 在希望放在同一子菜单中的第一组选项中的第一个 option 元素（参见前面的第 (8) 步）之前，输入 <optgroup。

(3) 输入 label="*submenutitle*">，这里的 *submenutitle* 是子菜单的标题。

(4) 在该组的最后一个 option 元素之后，输入 </optgroup>。

(5) 对每个子菜单重复第 (2) 步至第 (4) 步。

```
<label for="referral">Where did you find out
→ about us?</label>
<select id="referral" name="referral">
    <optgroup label="Online">
        <option value="social_network">Social
        → Network</option>
        <option value="search_engine">Search
        → Engine</option>
    </optgroup>
    <optgroup label="Offline">
        <option value="postcard">Postcard
        → </option>
        <option value="word_of_mouth">Word of
        → Mouth</option>
    </optgroup>
</select>
```

图 16.12.4　每个子菜单都有一个标题（在 optgroup 开始标签的 label 属性中指定）和一系列选项（使用 option 元素和常规文本定义）

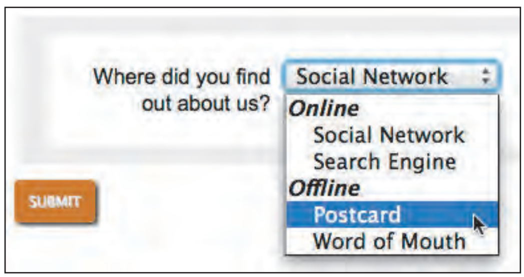


图 16.12.5　浏览器通常会对 optgroup 中的 option 缩进，从而将它们和 optgroup label 属性文本区别开

提 示　如果添加了 size 属性，那么选择框看起来会更像一个列表，且没有自动选中的选项，参见图 16.12.6（除非设置了 selected）。

提示　如果 size 大于选项的数量，访问者就可以通过点击空白区域让所有的选项处于未选中状态。

提示　可以对 option 元素添加 label 属性，该属性用于指定需要显示在菜单中的文本（替代了 option 标签之间的文本），参见 16.10 节。不过，Firefox 并不支持这一属性，因此最好不要用它。

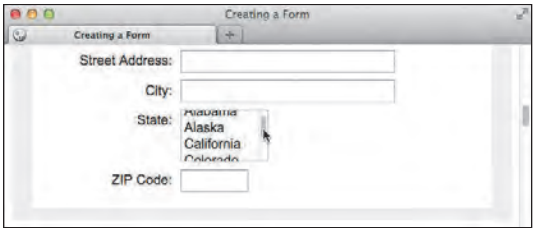


图 16.12.6　由于设置了 size 属性，菜单显示为一个有滚动条的列表，默认情况下没有选中任何选项。这个示例的代码为 <select id="state" name="state" size="3">，这让菜单的高度为三行

16.13　让访问者上传文件

有时需要让网站的用户向服务器上传文件（如照片、简历等），参见图 16.13.1。

```
<form method="post" action="show-data.php"
enctype="multipart/form-data">
    ...
    <label for="picture">Picture:</label>
    <input type="file" id="picture"
    → name="picture" />
    <p class="instructions">Maximum size of
    → 700k. JPG, GIF or PNG.</p>
    ...
</form>
```

图 16.13.1　要让访问者能够上传文件，必须正确地设置 enctype 属性，创建 input type="file" 元素。对 input 使用 multiple 属性可以允许上传多个文件（这里并没有包含该属性）。这是 HTML5 中新增的内容，它也得到了浏览器的广泛支持，不过，移动端浏览器和 IE 会直接忽略它（仅 IE10 支持）

处理文件上传

处理文件上传需要一些特殊的代码。可以在网上搜索“file upload script”（文件上传脚本）查看相关的资源。同时，服务器需要配置正确才能存储文件。如需帮助，可以联系网站托管商。

让访问者上传文件的步骤

(1) 输入 <form method="post" enctype="multipart/form-data"。enctype 属性可以确保文件采用正确的格式上传。

(2) 接下来，输入 action="*upload.url*">，其中的 *upload.url* 是处理上传文件的脚本的 URL。

(3) 为文件上传区域输入标签，以便访问者知道上传什么文件（参见 16.6 节）。

(4) 输入 <input type="file"，创建一个文件上传框和一个 Browse（浏览）按钮，如图 16.13.2 所示。

(5) 输入 id="*idlabel*"，这里的 *idlabel* 跟第 (3) 步中 label 的 for 属性值相同。

(6) 输入 name="*dataname*"，这里的 *dataname* 用于识别将要上传的文件。

(7) 如果需要，输入 size="*n*"，这里的 *n* 是访问者可以输入路径和文件名的字段的宽度。

(8) 如果需要，输入 multiple 或者 multiple="multiple"（HTML5 中这两种方式均可），这表示访问者可以上传一个以上的文件，参见图 16.13.1 的标题说明。

(9) 输入 > 或者 />。

(10) 跟平常一样结束表单，包括创建提交按钮，输入 </form> 结束标签。

16

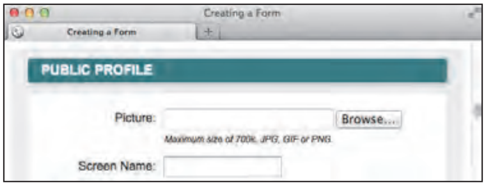


图 16.13.2　文件上传区域为用户提供了从其系统中选择文件的方式。对于 type="file" 的 input 元素，浏览器会自动创建 Browse（浏览）按钮。Chrome 和 Safari 不会创建框，它们只显示按钮。浏览器通常不允许像对其他表单元素那样对此类 input 设置样式

提 示　对于允许上传的表单，不能使用 get 方法。

16.14　创建隐藏字段

隐藏字段可以用于存储表单中的数据，但它不会显示给访问者（参见图 16.14.1 和图 16.14.2）。可以认为它们是不可见的文本框。它们通常用于存储先前的表单收集的信息，以便将这些信息同当前表单的数据一起交给脚本进行处理，如图 16.14.2 所示。

```
<form method="post" action="your-script.php">
    <input type="hidden" name="step"
    → value="6" />

    ... 其他表单字段 ...

    <input type="submit"
    → value="Submit Form" />
</form>
```

图 16.14.1　访问者不会看到这个输入框，但他们提交表单的时候，名“step”和值“6”会随着表单中从访问者输入获取的数据一起传送给服务器

创建隐藏字段的步骤

(1) 输入 <input type="hidden"。

(2) 输入 name="*dataname*"，这里的 *dataname* 确定要提交给服务器的信息。

(3) 输入 value="*data*"，这里的 *data* 是要提交的信息本身。它通常是表单处理脚本中的一个变量（参见图 16.14.1）。

(4) 输入 > 或者 />（在 HTML5 中两种方式均可）。

```
<form method="post" action="your-script.php">
    <input type="hidden" name="email"
    → value="<?= $email ?>" />

    ... 其他表单字段 ...

    <input type="submit"
    → value="Submit Form" />
</form>
```

图 16.14.2　创建隐藏字段时，可以使用脚本中的变量将字段的值设置为访问者原来输入的值（这个示例使用了 PHP 语法）

什么时候使用隐藏字段

这里有个隐藏字段非常方便的例子。假设你有一个表单，希望让访问者在提交表单之前有机会检查他们输入的内容。处理表单的脚本可以向访问者显示提交的数据，同时创建一个表单，其中有包含同样数据的隐藏字段。如果访问者希望编辑数据，他们只需后退就可以了。如果他们想提交表单，由于隐藏字段已经将数据填好了，因此他们就不需要再次输入数据了。

提 示　隐藏字段出现在表单标记中的位置并不重要，因为它们在浏览器中是不可见的。

提 示　不要将密码、信用卡号等敏感信息放到隐藏字段中。即便它们不会显示到网页中，访问者也可以通过查看 HTML 源代码看到它们（参见 2.8 节）。

> **提 示** 要创建访问者可见但不可修改的表单元素，有两种方法。一种是使用 disabled（禁用）属性（参见 16.16 节）。另一种是使用 readonly（只读）属性。与禁用字段不同，只读字段可以获得焦点，访问者可以选择和复制里面的文本，但不能修改这些文本。它只能应用于文本输入框和文本区域，例如，<input type="text" id="coupon" name="coupon" value="FREE" readonly />。还可以使用 readonly="readonly" 这样的形式，结果是一样的。

16.15 创建提交按钮

访问者输入的信息如果不发送到服务器，就没什么用。应该总是为表单创建提交按钮，让访问者可以将信息交给你。提交按钮可能呈现为文本（参见图 16.15.1 ~ 图 16.15.3）、可能是图像（参见图 16.15.4 和图 16.15.5），也可能是两者的结合（参见图 16.15.6 和图 16.15.7）。

```
<input type="submit" value="Create Profile"
→ class="btn" />
```

图 16.15.1 如果不填写 name 属性，则提交按钮的名－值对就不会传给脚本。由于通常不需要这一信息，因此这里的做法是有益的

```
.btn {
     background-color: #da820a;
     border: none;
     border-radius: 4px;
     box-shadow: 2px 2px 2px #333;
     color: #fff;
     mmargin: 12px 0 0 26px;
     padding: 8px;
     text-shadow: 1px 1px 0px #777;
}
```

图 16.15.2 通过使用类，我为提交按钮应用了背景、字体格式和一些 CSS3 特性。因为是类，所以可以将该样式重用到其他按钮上

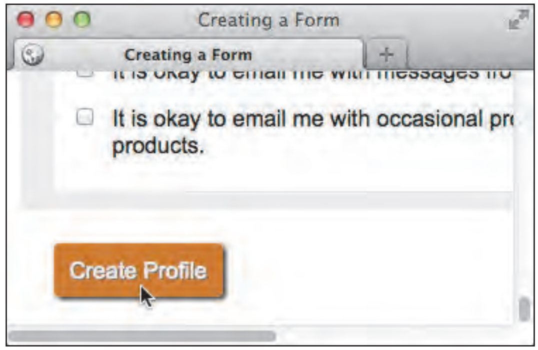


图 16.15.3 这里将按钮的 value 设为 Create Profile，比起默认值，这样的值对访问者来说含义更丰富（参见第一条提示）。如果激活提交按钮，就会将表单数据发给服务器上的脚本，从而可以利用这些信息

1. 创建提交按钮的步骤

(1) 输入 <input type="submit"。

(2) 如果需要，输入 value="*submit message*"，这里的 *submit message* 是将要出现在按钮上的文本。

(3) 最后输入 > 或者 />。

2. 创建带图像的提交按钮

有时设计师会创建一些超出 CSS3 能力（如渐变、阴影、圆角等）的按钮样式。这时，可以使用仅包含图像的 input 元素来提交表单（参见图 16.15.4 和图 16.15.5）。

(1) 创建 PNG、GIF 或者 JPEG 图像，通常最好是 PNG，因为文件尺寸小。

(2) 输入 <input type="image">。

(3) 输入 src="*image.url*"，其中的 *image.url* 是图像在服务器上的位置。

(4) 输入 alt="*description*"，这里的 *description* 是当图像无法显示时需要出现的文本。

(5) 输入 > 或者 /> 以结束图像提交按钮。

```
<input type="image" src="button-submit.png"
→ width="188" height="95" alt="Create Profile"
→ />
```

图 16.15.4　利用 type="image" 可以创建图像提交按钮，width 和 height 属性是可选的

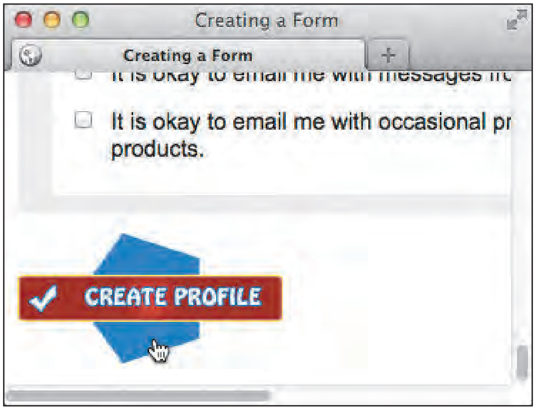


图 16.15.5　当用户将鼠标悬停在图像提交按钮上时，浏览器会将箭头替换为手形图标

3. 创建结合文本和图像的提交按钮

使用 button 元素可以创建包含其他 HTML 元素（而并非仅包含简单的文本值或图像）的按钮，参见图 16.15.6 和图 16.15.7。（如果要使用 button 元素，注意考虑 IE8 之前的 IE 的兼容性问题。更多细节可以在网上搜到。）

(1) 输入 <button type="submit">。

(2) 输入将要出现在按钮中图像左侧的文本（如果有的话）。

(3) 输入 "*image.url*"，这里的 *image.url* 是出现在按钮上的图像名称。

(4) 输入 alt="*alternate text*"，这里的 *alternate text* 是当图像无法显示时需要出现的文本。

(5) 如果需要，添加其他图像属性。

(6) 输入 > 或者 /> 以结束图像。

(7) 输入将要出现在按钮中图像右侧的文本（如果有的话）。

(8) 输入 </button>。

```
<button type="submit" class="btn">
→ <img src="check.png" width="21" height="21"
→ alt="" /> Create Profile</button>
```

图 16.15.6　button 元素可以为提交按钮的内容提供更大的灵活性。这个元素既包含图像，也包含文本。button 元素还可以包含其他的 HTML 元素

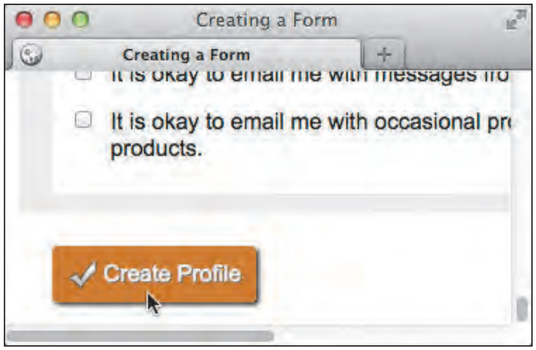


图 16.15.7　除去那个对勾的图像，这个按钮与图 16.15.3 中的按钮很像，因为它们都有 .btn 类（参见图 16.15.2）。当鼠标移到这个按钮上时，指针会显示为一个箭头（参见图 16.15.3）

提 示　如果省略 value 属性，那么根据不同的浏览器，提交按钮就会显示默认的 Submit 或者 Submit Query。

提 示　如果有多个提交按钮，可以为每个按钮设置 name 属性和 value 属性，从而让脚本知道用户按下的是哪个按钮。否则，最好省略 name 属性。

提 示　可以使用 button 元素创建不包含图像的提交按钮。任何时候，如果你的表单需要一个以上的提交按钮，就应避免使用 button 元素，因为针对这种情况，不同浏览器的行为并不完全一致。

提 示 表单还可以有重置按钮，用于将表单数据还原为页面加载时（在访问者填写表单之前）的样子。创建重置按钮可以使用 <input type="reset" /> 或 <button type="reset">Reset</button>。我们也可以为重置按钮添加样式。

提 示 HTML5 对 type="email" 和 type="URL" 的 input 添加了自动验证功能。对提交按钮使用 formnovalidate 属性可以关闭该功能，如 <input type="submit" formnovalidate />。

16.16 禁用表单元素

在某些情况下，你可能不想让访问者使用表单中的某些部分。例如，你可能希望在所有必填字段完成之前禁用提交按钮，如图 16.16.1 所示。

禁用表单元素的值不会发送到服务器，如果使用键盘在页面中导航，它也会被跳过。


```
...
    <div id="choices">
        <p>
            <input type="radio" name="how" value="advertisement" id="advertisement" />
            <label for="advertisement">Advertisement</label>
        </p>
        ...
        <p>
            <input type="radio" name="how" value="other" id="other" />
            <label for="other">Other</label>
        </p>
        <p>
            <textarea id="other-description" cols="35" rows="5" placeholder="TV, school, bingo game,
            ➝ etc." title="Please describe how you heard about us." disabled="disabled"></textarea>
        </p>

        <input type="submit" value="Submit" class="btn" />
    </div>
...
</form>

<!-- 这块要放到最后，刚好在</body>之前 -->
<script src="js/toggle-textarea.js"></script>
</body>
</html>
```

图 16.16.1 这里对 textarea 元素添加了 disabled 属性，并在页面最底部引入了一个 JavaScript 文件。该文件包含了一段脚本，其作用是当用户选择 Other（其他）单选按钮时，让 textarea 变为可用的（如图 16.16.2 和图 16.16.3 所示）。选择其余两个单选按钮中的任意一个，则会禁用 textarea（注意，在实践中，我会使用 JavaScript 添加 disabled 属性，而不是像这样写在 HTML 中。这样，禁用了 JavaScript 的访问者仍可填写该文本区域。第 19 章的 toggle-textarea.js 就实现了这样的功能）

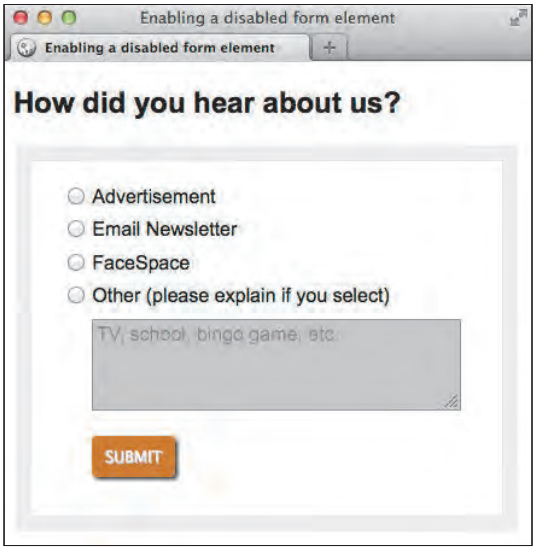


图 16.16.2　当 Other 单选按钮未选中时，文本区域是灰色的、被禁用的，用户无法选中该框并输入内容

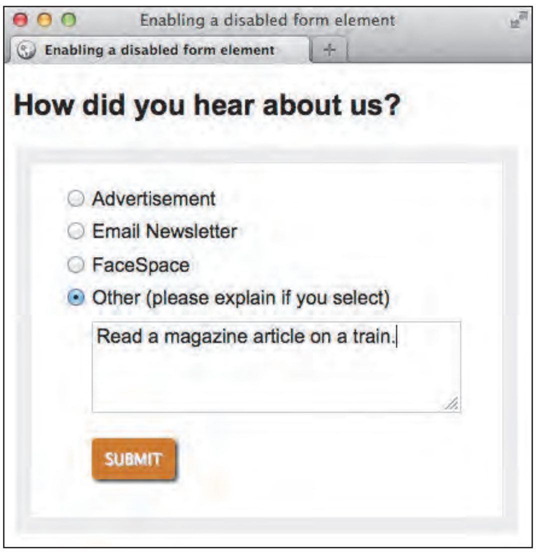


图 16.16.3　由于使用了 JavaScript，当访问者选择 Other 单选按钮以后，文本区域变成了白色，用户可以输入文本以提交到服务器

禁用表单元素的方法

在表单元素的开始标签后输入 disabled 或者 disabled="disabled"（两种方法在 HTML5 中均可以）。

提示　可以使用 JavaScript 将表单元素的状态由禁用改为可用（也可以反过来）。对 JavaScript 的讲解超出了本书的范围，但我在第 19 章提供了图 16.16.1 引用的 toggle-textarea.js 文件（参见 19.1 节）。该文件中有一些用于解释其工作原理的注释。要学习 JavaScript，eloquentjavascript.net 是个不错的资源。

提示　更多关于 script 元素的信息参见第 19 章。

提示　要理解 disabled 和 readonly 属性的区别，参见 16.14 节最后一条提示。

16.17　根据状态为表单设置样式

有时，你可能需要根据表单的状态或是否必须包含某个属性的情况设置不同的样式。例如，你可能需要将必填字段与其他字段在样式上区分开。

你可能会想起本书其他地方（如 9.7 节）介绍过的伪类。不过，CSS 提供了其他一些根据表单元素状态设置其样式的伪类。其中的大多数都是 CSS3 新增的。我总结了其中得到浏览器广泛支持的一些，如表 16.17.1 所示。其他的参见 Peter-Paul Koch 的列表——打开 www.quirksmode.org/css/selectors/ 并拖到 UI state pseudo-classes（UI 状态伪类）部分。

表 16.17.1　伪类

选　择　器	应　　用	浏览器支持情况
:focus	获得焦点的字段（参见图 16.17.1 和图 16.17.2）	IE8+ 及其他
:checked	选中的单选按钮或复选框（参见图 16.17.3 和图 16.17.4）	IE9+ 及其他
:disabled	具有 disabled 属性的字段（参见图 16.17.5）	IE9+ 及其他
:enable	与 :disabled 相反	IE9+ 及其他
:required	具有 required 属性的字段（参见图 16.17.6 和图 16.17.7）	IE10+、Safari 5+ 及其他
:optional	与 :required 相反	IE10+、Safari 5+ 及其他
:invalid	其值与 pattern 属性给出的模式不匹配的字段；或值不是有效电子邮件格式的电子邮件框，值不是有效 URL 格式的 URL 框，以及任何标记为 required 但值为空的字段（参见提示），如图 16.17.8 和图 16.17.9 所示	IE10+、Safari 5+ 及其他
:valid	与 :invalid 相反	IE10+、Safari 5+ 及其他

注意：“IE9+ 及其他”指的是 Internet Explorer 9 及之后的版本支持该选择器，其他现代浏览器也支持该选择器。其他的说明也是类似的


```
input:focus,
textarea:focus {
     background-color: greenyellow;
}
```

图 16.17.1　这条规则为任意获得焦点的 input（包括提交按钮）或 textarea 添加了背景色。要定位特定的 input 类型，可以包含一个属性选择器，如 input[type="submit"]:focus { backgroundcolor: #ff8c00; } 就仅为获得焦点的提交按钮设置样式

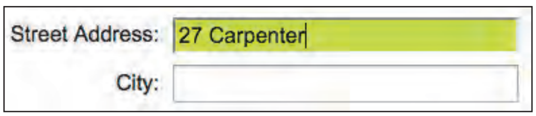


图 16.17.2　第一个字段获得了焦点，因此它的背景是浅绿色的（虽然并不是最舒服的颜色，但能说明问题）

```
input:checked + label {
     color: green;
}
```

图 16.17.3　这条规则为选中的单选按钮或复选框（或指定了 checked 属性的单选按钮或复选框）添加样式，效果如图 16.17.4 所示

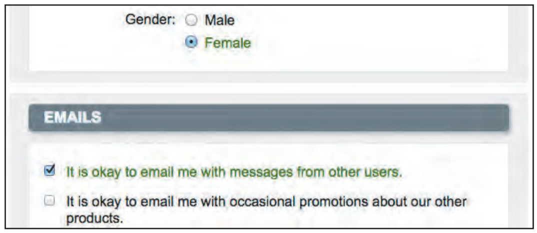


图 16.17.4　只有选中的单选按钮和复选框是绿色的

```
textarea:disabled {
     background-color: #ccc;
     border-color: #999;
     color: #666;
}
```

图 16.17.5　这是用于图 16.16.2 中代码的样式规则

```
input:required,
textarea:required {
     border: 2px solid #000;
}
```

图 16.17.6　所有必填的 input 和 textarea 元素会有一个更醒目的边框

针对特定状态设置表单元素样式的步骤

(1) 输入 selector，这里的 selector 包含

表 16.17.1 中显示的一种或多种状态，接着输入 { 开始样式规则的声明。

(2) 输入你想为该状态设置的任何 property: value; 声明。关于 CSS 属性的讲解是从第 8 章开始的。

(3) 输入 } 结束样式规则的声明。



图 16.17.7　如你所见，只有 First Name（名）字段是必填的。红色的星号并不是图 16.17.6 中的 CSS 产生的效果。对于必填的字段，在其标签中添加星号或“必填”字样是一种常见的做法，因此这里添加了星号。这样，不支持 :required 的浏览器也会表现出这是一个必填字段的样子

```
input[type="email"]:invalid {
     color: red;
}

input[type="email"]:valid {
     color: black;
}
```

图 16.17.8　电子邮件框中的值如果不是有效的电子邮件地址，该框中的字就会变成红色的

提 示　页面一开始加载就会应用 :invalid 状态，因此，根据你所设置的状态，有可能会出现一些意外的结果。例如，如果规则是 input:invalid { background-color: pink; }，那么必填的 input 字段在用户填写任何内容之前就会显示粉红色的背景。（任何标记为 required 但值为空的字段会被认为处于无效状态。）为了将必填字段排除在该规则之外，可以使用 :not 伪类，如 input:invalid:not(:required) { border: 2px solid red; }。除了 IE9 之前的版本，所有浏览器都支持 :not 伪类。

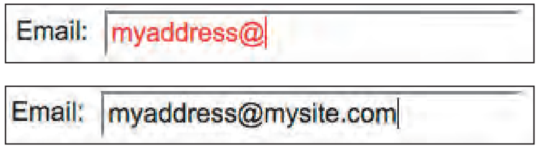


图 16.17.9　输入的内容都是红色的，直到输完一个有效的电子邮件地址（此时输入的内容会变成黑色的）

提 示　上一条提示讲到的情境还有一种应对方法，即提交表单时，使用 JavaScript 为 form 元素添加一个类，并在样式表中为该类创建无效样式，例如，.submitted input:invalid { background-color: red; }。示例代码及其他提示参见 Peter Gasston 的文章（位于 html5doctor.com/css3-pseudo-classes-and-html5-forms）。

提 示　对于浏览器无法识别的伪类，浏览器会直接忽略其中的样式规则。对于 Web 设计人员和开发人员来说，这通常是可以接受的——使用功能较强的浏览器的访问者获得增强的体验。Keith Clark 的 Selectivzr（http://selectivizr.com/）是一个用于让旧浏览器理解这些选择器的 JavaScript 文件。

使用属性选择器为表单设置样式

别忘了，还可以使用属性选择器定位拥有特定属性的表单字段，例如：

- [autocomplete]
- [autofocus]
- [multiple]（仅限于电子邮件框和文件上传框）
- [placeholder]
- [type="email"]（参见图 16.17.8）、[type="url"] 等（其他的输入框类型都可以像这样使用）

第 17 章

视频、音频和其他多媒体

17

本章内容

- ❑ 第三方插件与步入原生
- ❑ 视频文件格式
- ❑ 在网页中添加单个视频
- ❑ 为视频添加控件和自动播放
- ❑ 为视频指定循环播放和海报图像
- ❑ 阻止预加载视频
- ❑ 使用多种来源的视频和备用文本
- ❑ 提供可访问性
- ❑ 音频文件格式
- ❑ 为网页添加带控件的音频文件
- ❑ 自动播放、循环和预加载音频
- ❑ 提供带有备用设置的多种音频来源
- ❑ 添加具有备用 Flash 的视频和音频
- ❑ 高级多媒体
- ❑ 更多资源

在页面上添加电影、声音、图像和动画可以增强访问者的体验。在 HTML5 出现之前，为网页添加多媒体的唯一办法就是使用第三方的插件（如 Adobe Flash Player 和苹果的 QuickTime）。通过引入原生的多媒体（浏览器负责一切），HTML5 改变了这一状况。

并非所有支持 HTML5 的浏览器都支持相同格式的视频和音频格式。你将在本章学到如何提供多种格式（以及为不支持任何一种格式的浏览器提供备用 Flash），让不同的访问者都能正常使用。

注意，本章的目的是介绍多媒体 Web 文件，重点放在了所需的 HTML5 代码上。这里不会讲解如何创建多媒体内容，只会讲解如何让访问者可以浏览它们。

17.1　第三方插件和步入原生

如上文所述，在 HTML5 出现之前，可以通过第三方插件为网页添加音频和视频，但这样做有一些问题。在某个浏览器中嵌入 Flash 视频的代码在另一个浏览器中可能不起作用，也没有包围它的优雅方式。而且，这样做会影响用户访问网站的体验，因为像 Flash 这样的插件会占用大量的计算资源。有时，浏览器会变慢，甚至崩溃。

考虑到这些问题，HTML5 规范中添加了原生的多媒体。这样做有很多好处：速度更快（任何浏览器原生的功能势必比插件要快一些），媒体播放按钮和其他控件内置到浏览器，对插件的依赖极大地降低（但也并非完全不依赖，这在后面将会看到）。

对于任何一套标准，都有关于 HTML5 原生多媒体及其支持的文件格式的问题。最初，HTML5 规范指定了两种兼容 HTML5 的浏览器必须支持的媒体格式（分别对应音频和视频）。这本来是很好的，但并非所有的厂商都愿意遵循。这意味着你需要为你的媒体提供一种以上的格式才能让它在兼容 HTML5 的浏览

器中播放。后面我们将详细地讨论这些问题。

当苹果宣布不再在其移动设备（包括 iPhone 和 iPad）上支持 Flash 时，HTML5 及原生媒体就变得更为实用了。随着这些设备的日渐普及，过去播放媒体文件对 Flash 的依赖正在迅速削减，因此迫切需要一种新的解决方案。于是，HTML5 原生多媒体进入了这一领域并展现出了强大的力量，因为苹果移动设备的浏览器是支持 HTML5 的。其他移动设备也迅速跟进。

数字版权管理（DRM）

在嵌入音频和视频文件的过程中，你一定注意到了这样一个事实——所有人都看得见指向源文件的 URL，从而利用它下载并“盗取”你的内容，就像嵌入的图像以及 HTML、JavaScript 和 CSS 源文件一样。对此我们毫无办法。

尽管已经有了一些相关的讨论，但 HTML5 并没有提供任何保护媒体内容的方法。因此，如果你很在意对媒体文件的保护，那么暂时不要使用 HTML5 原生多媒体。

17.2　视频文件格式

HTML5 支持三种视频文件格式（即编解码器）。

- Ogg Theora 使用的文件扩展名为 .ogg 或 .ogv，支持它的浏览器包括 Firefox 3.5+、Chrome 4+、Opera 10.5+ 以及 Android 版 Firefox。
- MP4（H.264）使用的文件扩展名为 .mp4 或 .m4v，支持它的浏览器包括 Safari 3.2+、Chrome 4-?（参见提示）、Internet Explorer 9+、iOS（Mobile Safari）和 Android 2.1+、Android 版 Chrome、Android 版 Firefox 和 Opera Mobile 11+。
- WebM 使用的文件扩展名为 .webm，支持它的浏览器包括 Firefox 4+、Chrome 6+、Opera 10.6+、Android 2.3+、Android 版 Chrome、Android 版 Firefox 和 Opera Mobile 14。

什么是编解码器

编解码器是使用压缩算法对数据的数字流进行编码和解码，使之更适合播放的计算机程序。

编解码器的目标通常是在保证音频和视频所能达到的最高质量的情况下减小文件尺寸。

当然，不同编解码器的表现是不一致的。

设置 MIME 类型

在有的浏览器中，如果没有设置正确的 MIME 类型，媒体文件就不会播放。如果你的网站运行在 Apache Web 服务器上（很可能是它），可以通过 .htaccess 文件设置 MIME 类型。它是一个文本文件，通常与主页一起位于网站的根目录。

这是可以在 .htaccess 文件中添加的 MIME 类型（可以使用文本编辑器编辑该文件）。

```
AddType video/ogg .ogv
AddType video/mp4 .mp4
AddType video/webm .webm
AddType audio/ogg .ogg
AddType audio/mp3 .mp3
```

如果你的网站已经拥有该文件，将其命名为 a.htaccess，从 Web 服务器上下载下来，添加上述 MIME 类型，再上传到服务器上，重新命名为 .htaccess。如果你的网站没有该文件，可以重新创建一个。

对于更新 .htaccess，如有问题，请联系 Web 托管商。

转换文件格式

如果你已经拥有一个视频资源并希望将它转换为其他的文件格式，有大量的工具可以帮你完成这一任务。下面就是两个这样的工具：

- Miro Video Converter（www.mirovideo-converter.com）
- HandBrake（http://handbrake.fr）

提示 开发者至少需要为视频提供两种格式（MP4 和 WebM），才能确保获得所有兼容 HTML5 的浏览器的支持。

提示 Google 曾经说过在 Chrome 中放弃对 MP4 的支持，不过他们没这么做。Firefox 正在逐步提供对 MP4 的支持。在桌面端，Windows 7+ 首次提供了对它的支持，不过需要用户在机器上安装相应的编解码器。

提示 如果用户的机器上安装了 WebM，那么 WebM 在 IE 9+ 或 Safari 中都可以正常运行。

17.3 在网页中添加单个视频

要在 HTML5 网页中添加视频，需要使用新的 video 元素。这一过程再简单不过了，如图 17.3.1 所示。浏览器会计算视频的尺寸，并在加载文件后以计算的尺寸显示视频，参见图 17.3.2。开发人员也可以自行设定视频尺寸，如图 17.3.3 所示。

如果浏览器不能识别你指定的视频格式，就无法显示该视频，参见图 17.3.4。

```
...
<body>
    <video src="paddle-steamer.webm"></video>
</body>
</html>
```

图 17.3.1 指定单个 WebM 视频（不含控件）

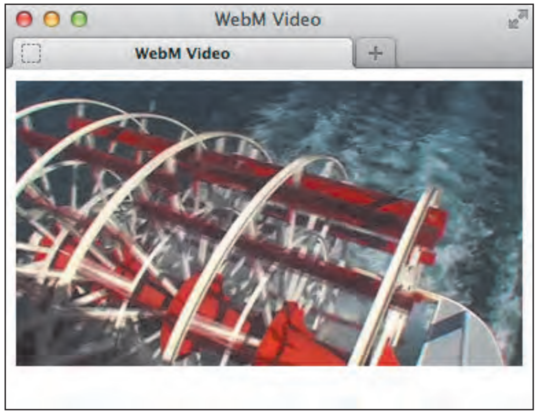


图 17.3.2 默认情况下，视频会暂停在第一帧，目前我们得到的视频没有播放按钮，也就是不能观看！下一节会纠正这个问题

```
...
<body>
    <video src="paddle-steamer.webm"
    ➝ width="369" height="208"></video>
</body>
</html>
```

图 17.3.3 这里为视频设置的尺寸符合其正常大小，因此其效果与图 17.3.3 相同。不过，同图像一样，可以使用其他大小的 width 和 height 值，浏览器会对视频进行缩放，并尽可能地保证质量

17

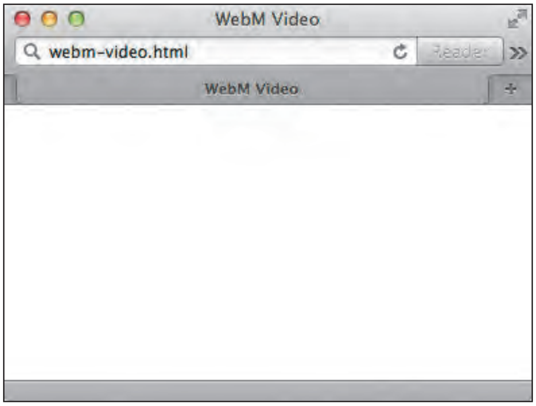


图 17.3.4　像 Safari 这样不支持 WebM 的浏览器会显示一片空白。这并不好！后面会介绍如何指定多种视频格式，从而支持各种的浏览器

1. 在网页中添加单个视频的步骤

(1) 获取视频资源。

(2) 输入 <video src="*my-video.ext*"></video>，这里的 *my-video.ext* 是视频文件的位置、名称和扩展名。就这么简单（参见图 17.3.2 和图 17.3.4）！

2. video 属性

除了 src，还有哪些属性可以用在 video 元素上呢？表 17.3.1 列出了这些属性。其中的某些属性为视频提供了很高的灵活性。

表 17.3.1　video 属性

属　　性	描　　述
src（源）	指定视频文件的 URL
autoplay（自动播放）	当视频可以播放时立即开始播放
controls（控件）	添加浏览器为视频设置的默认控件
muted（静音）	让视频静音
loop（循环）	让视频循环播放
poster（海报）	指定视频加载时要显示的图像（而不显示视频的第一帧）。接受所需图像文件的 URL
width（宽度）	视频的宽度（以像素为单位），通常默认为 300
height（高度）	视频的高度（以像素为单位），通常默认为 150
preload（预加载）	告诉浏览器要加载的视频内容的多少。可以是以下三个值： ❑ none 表示不加载任何视频 ❑ metadata 表示仅加载视频的元数据（如长度、尺寸等） ❑ auto 表示让浏览器决定怎样做（这是默认的设置）

17.4　为视频添加控件和自动播放

目前仅向你展示了在网页中添加视频的最简单的方法，而示例中的视频甚至不会自动开始播放，因为我们没有让它这样做。此外，访问者也不能开始播放视频，因为播放器没有显示任何控件。

很容易就可以通过图 17.4.1 中的代码改变这种情况。controls 属性会告诉浏览器添加一套用于控制视频播放的控件。每个浏览器都有自己默认的控件，它们看起来都不一样（参见图 17.4.2 ～图 17.4.6）。

通常，只有用户点击播放按钮后，视频才会开始播放。可以添加 autoplay 属性，让视频自动播放（如图 17.4.7 所示）。

```
...
<body>
    <video src="paddle-steamer.webm"
    → width="369" height="208" controls>
    → </video>
</body>
</html>
```

图 17.4.1　添加单个 WebM 视频（这次是包含控件的）



图 17.4.2　Firefox 中的视频控件。从这里可以看到 Firefox 中的视频比其他浏览器中的视频长 1 秒



图 17.4.3　Chrome 中的视频控件



图 17.4.4　Internet Explorer 10 中的视频控件



图 17.4.5　Opera 中的视频控件

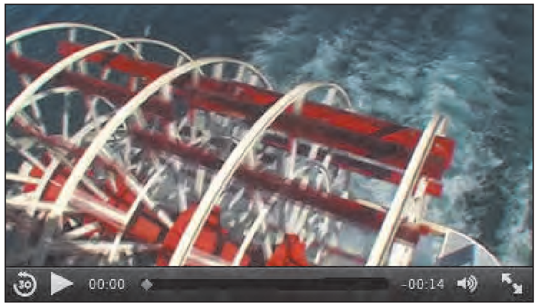


图 17.4.6　Safari 中的视频控件

```
...
<body>
    <video src="paddle-steamer.webm"
    → width="369" height="208" autoplay
    → controls></video>
</body>
</html>
```

图 17.4.7　现在，video 元素有了表 17.3.1 中的三个属性。由于有 autoplay，因此视频会自动播放。由于有 controls，访问者有了暂停按钮。这些属性出现的顺序无关紧要

1. 为视频添加控件

输入 `<video src="my-video.ext" controls></video>`，其中 *my-video.ext* 指向你的视频文件。

2. 为视频添加自动播放

输入 `<video src="my-video.ext" autoplay controls></video>`，这里的 *my-video.ext* 指向你的视频文件。

17

如果浏览器不支持某种视频格式

如果查看代码示例（参见图 17.4.1 和图 17.4.7）的浏览器不支持你使用的视频文件格式，它会显示视频的控件条，或者显示一个空的白色矩形区域（大多数情况下是这样），或者显示海报图像（前提是通过 poster 属性指定了海报）。

例如，Internet Explorer 和 Safari 不支持 WebM，但支持 MP4。为了在图 17.4.4 和图 17.4.6 中演示它们的播放控件，我创建了同时使用 MP4 和 WebM 格式的页面。关于实现方法，参见 17.7 节。

如果视频没有通过 width 和 height 指定尺寸，空的矩形通常为 300 像素 ×150 像素大小。如果包含了 controls，但使用了一种 IE10 不支持的格式，它会显示一个空白的矩形，以及 Invalid Source（无效来源）的字样。

布尔类型的属性

回想第 1 章，你应该能记得，布尔类型的属性（如 controls 和 autoplay）不需要指定值，因为它们是在几个值之间进行选择。无论采取哪种方式，它们出现在 video 和 audio 元素中产生的效果是一样的。

本书的示例都没有为这些布尔类型的属性指定值，不过，图 17.4.7 中的代码也可以写作：

```
<video src="paddle-steamer.webm"
controls="controls" autoplay="autoplay">
</video>.
```

17.5　为视频指定循环播放和海报图像

不仅可以将视频设为自动播放，还可以将它设为持续播放，直到停止，如图 17.5.1 所示。（不过这一做法并不推荐，考虑一下用户的感受就知道原因了。）要实现循环播放，只需要使用 autoplay 和 loop 属性。

如果不设置 autoplay 属性，通常浏览器会在视频加载时显示视频的第一帧。你可能想对此作出修改，指定你自己的图像。这可以通过海报图像实现，参见图 17.5.2 和图 17.5.3。

```
...
<body>
    <video src="paddle-steamer.webm"
    ➝ width="369" height="208" autoplay
    ➝ loop></video>
</body>
</html>
```

图 17.5.1　设置了自动播放和循环播放的单个 WebM 视频。如果这里不设置 controls，访问者就无法停止视频！因此，如果将视频指定为循环，最好包含 controls。即便如此，自动循环播放的视频也可能让访问者非常不爽

```
...
<body>
    <video src="paddle-steamer.webm"
    ➝ width="369" height="208"
    ➝ poster="paddle-steamer-poster.jpg"
    ➝ controls></video>
</body>
</html>
```

图 17.5.2　指定了海报图像（当页面加载并显示视频时显示该图像）的单个 WebM 视频（含控件）

1. 为视频添加自动播放和循环播放

输入 <video src="*my-video.ext*" autoplay loop></video>，这里的 *my-video.ext* 指向你的视频文件。

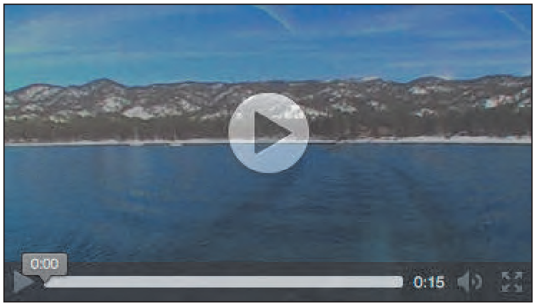


图 17.5.3 显示海报图像的视频。在这个例子中，图像是取自视频本身的一幅屏幕截图

2. 为视频指定海报图像

输入 `<video src="my-video.ext" controls poster="my-poster.jpg"></video>`，其中，*my-video.ext* 指向你的视频文件，*my-poster.jpg* 是想用做海报图像的图像。

17.6 阻止视频预加载

如果认为用户观看视频的可能性较低（如该视频并不是页面的主要内容），那么可以告诉浏览器不要预先加载该视频，参见图 17.6.1。这样能节省带宽，尤其对于移动设备用户来说好处多多。

对于设置了 preload="none" 的视频，在初始化视频之前，浏览器显示视频的方式并不一样（参见图 17.6.2 和图 17.6.3）。

```
...
<body>
    <video src="paddle-steamer.webm"
    ➝ preload="none" controls></video>
</body>
</html>
```

图 17.6.1 页面完全加载时也不会加载的单个 WebM 视频。仅在用户试着播放该视频时才会加载它。注意我省略了 width 和 height 属性

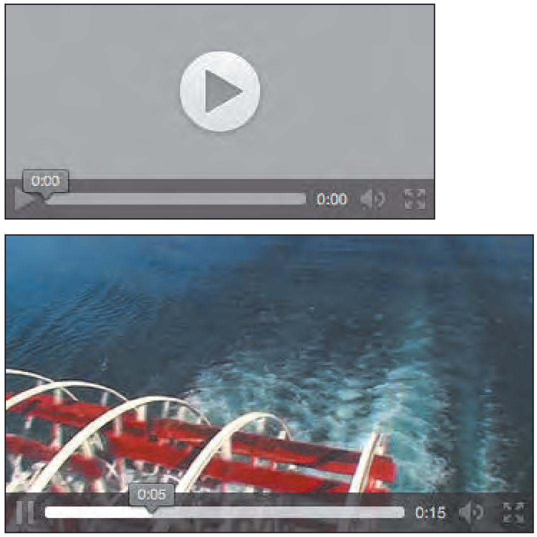


图 17.6.2 在 Firefox 中将 preload 设为 none 的视频。如你所见，什么也不会显示，因为浏览器没有得到关于该视频的任何信息（连尺寸都不知道），也没有指定海报图像。如果用户播放视频，浏览器会获取视频的尺寸，并调整视频大小



图 17.6.3 Chrome（上）在控制组件上面显示一个空白的矩形。这时，控制组件的大小比访问者播放视频时（下）显示的组件要窄一些

告诉浏览器不预先加载视频的步骤

输入 <video src="*my-video.ext*" preload="none" controls></video>，这里的 *my-video.ext* 指向你的视频文件。

> **其他预加载设置**
>
> preload 的默认值是 auto。这会让浏览器具有用户将要播放该视频的预期，从而做好准备，让视频可以很快进入播放状态。浏览器会预先加载大部分视频甚至整个视频。因此，在视频播放的过程中对其进行多次开始、暂停的操作会变得更不容易，因为浏览器总是试着下载较多的数据让访问者观看。
>
> 在 none 和 auto 之间有一个不错的中间值，即 preload="metadata"。这样做会让浏览器仅获取视频的基本信息，如尺寸、时长甚至一些关键的帧。在开始播放之前，浏览器不会显示白色的矩形，而且视频的尺寸也会与实际尺寸一致。
>
> 使用 metadata 会告诉浏览器，用户的连接速度并不快，因此需要在不妨碍播放的情况下尽可能地保留带宽资源。

17.7　使用多种来源的视频和备用文本

一切看起来都很棒，不过你也应该注意到了，前面所有的例子都只用了一个视频文件（同时意味着只有一种格式）。你已经知道了，要获得所有兼容 HTML5 的浏览器的支持，至少需要提供两种格式的视频：MP4 和 WebM。

如何做到呢？这时就要用到 HTML5 的 source 元素了。通常，source 元素用于定义一个以上的媒体元素（在这个例子中为 video）的来源。

一个 video 元素中可以包含任意数量的 source 元素，因此为我们的视频定义两种不同的格式是相当容易的，如图 17.7.1 所示。浏览器会加载第一个它支持的 source 元素引用的文件格式，并忽略其他的来源（如图 17.7.2 和图 17.7.3 所示）。无法播放 HTML5 视频的浏览器则会显示你提供的消息中的备用链接（如图 17.7.1 所示）。

```
…
<body>
    <video width="369" height="208" controls>
       <source src="paddle-steamer.mp4"
       ➝ type="video/mp4">
       <source src="paddle-steamer.webm"
       ➝ type="video/webm">
       <p><a href="paddle-steamer.mp4">
       ➝ Download the video</a></p>
    </video>
</body>
</html>
```

图 17.7.1　这里为视频定义了两个源：一个 MP4 文件和一个 WebM 文件。（注意 video 开始标签中没有像之前指定单个视频来源时使用的 src 属性。）旧的浏览器则只会显示 p 元素中的文本（参见图 17.7.4）。要确保备用链接或文本位于 video 里面，不然所有的浏览器都会显示它

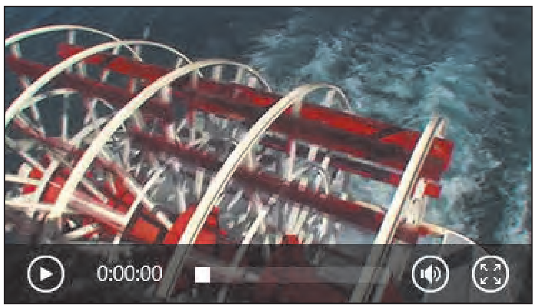


图 17.7.2　IE10 等支持 MP4 的文件会加载 paddle-steamer.mp4

1. 指定两种不同的视频来源（带有备用信息）

(1) 获取视频资源（这次需要两个）。

(2) 输入 <video controls> 开始 video 元素（含默认控件集）。

(3) 输入 <source src="*my-video.mp4*" type="video/mp4">，这里的 *my-video.mp4* 是 MP4 视频源文件的名称。

(4) 输入 <source src="*my-video.webm*"type="video/webm">，这里的 *my-video.webm* 是 WebM 视频源文件的名称。

(5) 为旧浏览器输入备用链接或者备用文本信息，参见图 17.7.4。

(6) 输入 </video> 结束 video 元素。

2. 多个媒体源的更多信息

接下来我们将学习 source 元素的各种可用的属性，不过在此之前，不妨先简单地了解一下为同一媒体指定多个来源的工作原理。

当浏览器发现 video 元素时，首先会查看该元素本身是否定义了 src。如果没有，就会检查 source 元素。浏览器会逐个查看这些来源，直到找到它可以播放的一个来源。一旦找到这样一个，就会播放它并忽略其他的来源。

在前一个例子中，Safari 会播放 MP4 文件（参见图 17.7.2）并完全忽略 WebM 文件，而 Firefox 则无法播放 MP4 文件，从而转向它能播放的 WebM 文件，如图 17.7.3 所示。（Firefox 正致力于将对 MP4 的支持添加到所有版本，因此当你读到本书的时候情况可能有所改变。）



图 17.7.3　不支持 MP4 但支持 WebM 的浏览器会加载 paddle-steamer.webm

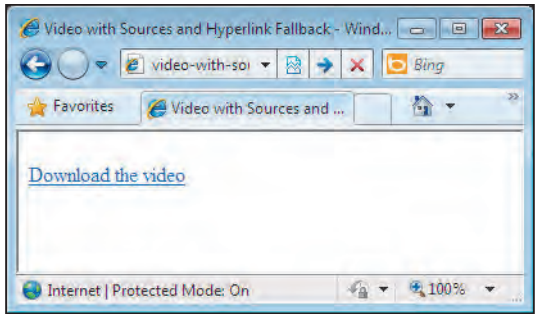


图 17.7.4　IE8 会忽略 video 和 source 元素，仅显示下载链接。这里我选择链接到视频的 MP4 版本（参见图 17.7.1），我也可以选择 WebM 版本，或者提供两个版本的链接

既不识别 video 元素又不识别 source 元素的浏览器（即不支持 HTML5 的浏览器）在解析文档时会完全忽略这些标签，它只会显示在 video 元素结束标签之前输入的文本。

表 17.7.1 列出了 source 元素的属性。

表 17.7.1　source 的属性

名　称	描　述
src	视频来源的 URL
type	用于指定视频的类型，帮助浏览器决定它是否能播放该视频。如图 17.7.1 所示，该属性的值反映的是视频的格式，即编解码器（如 video/mp4、video/webm、video/ogg）
media	用于为视频来源指定 CSS3 媒体查询，从而可以为具有不同屏幕尺寸的设备指定不同的（如更小的）视频

提示　图 17.7.1 中提供的备用内容除了可以放置链接，还可以放置视频的截图或其他内容，例如，`<p>Sorry, your browser doesn't support HTML5 video.</p>`，然后再提供一个链接。

提示　可以在 www.bigbuckbunny.org/index.php/download/ 找到一些免费的视频，用于试验 video 和 source 元素。该网站没有 WebM 格式的视频，不过你可以通过 17.2 节提供的工具进行格式转换。

17.8　提供可访问性

利用现代浏览器提供的原生可访问性支持，原生多媒体可以更好地使用键盘进行控制，这是原生多媒体的另一个好处。

HTML5 视频和音频的键盘可访问性支持在 Firefox、Internet Explorer 和 Opera 中表现良好。不过对于 Chrome 和 Safari，截至本书写作之际，实现键盘可访问性的唯一办法是自制播放控件。为此，需要使用 JavaScript Media API（这也是 HTML5 的一部分），不过这已经超出了本章的讨论范围。

HTML5 还指定了一种新的文件格式 WebVTT（Web Video Text Track，Web 视频文本轨道）用于包含文本字幕、标题、描述、篇章等视频内容。关于 WebVTT 和字幕的进一步讨论已经超出了本章的范围，不过更多信息可以参见 www.iandevlin.com/blog/2011/05/html5/webvtt-and-video-subtitles，其中包括为了对接规范修改在 2012 年进行的更新。

提示　Ian Devlin 的 *HTML5 Multimedia: Develop and Design*（Peachpit Press，2011）一书中有专门演示如何创建个人的可访问控件集和使用 WebVTT 的章节。http://net.tutsplus.com/tutorials/html-csstechniques/an-in-depth-overview-of-html5-multimedia-and-accessibility/ 摘录了本书部分章节。

提示　Terrill Thompson 比较了不同浏览器对 HTML5 视频可访问性的支持情况，见 http://terrillthompson.com/blog/366。

提示　WebVTT 规范还在制订之中，因此通过它设置标题等内容的方法未来有可能会变化。

17.9　音频文件格式

学会了使用 HTML5 原生媒体在网页中添加视频之后，下面看看如何添加音频。同 HTML5 视频一样，HTML5 也支持大量不同的音频文件格式（编解码器）。

- Ogg Vorbis 使用的文件扩展名为 .ogg，支持它的浏览器包括 Firefox 3.5+、Chrome 5+ 和 Opera 10.5+。
- MP3 使用的文件扩展名为 .mp3，支持它的浏览器包括 Safari 5+、Chrome 6+、Internet Explorer 9+ 和 iOS。
- WAV 使用的文件扩展名为 .wav，支持它的浏览器包括 Firefox 3.6+、Safari 5+、Chrome 8+ 和 Opera 10.5+。
- AAC 使用的文件扩展名为 .aac，支持它的浏览器包括 Safari 3+、Internet Explorer 9+、iOS 3+ 和 Android 2+。
- MP4 使用的文件扩展名为 .mp4，支持它的浏览器包括 Safari 3+、Chrome 5+、Internet Explorer 9+、iOS 3+ 和

Android 2+。

- Opus 使用的文件扩展名是 .opus。这是一种新的音频文件格式，在本书写作之时只有 Firefox 支持。

你应该还记得，MP4 是一种视频编解码器，不过它也可以仅对音频数据进行编码。

提 示　对于视频，需要使用两种不同的文件格式才能确保获得所有兼容 HTML5 的浏览器的支持。对于音频，最好的两种格式是 Ogg Vorbis 和 MP3。

提 示　在 17.2 节的补充材料“转换文件格式”中提到的 Miro Video Converter 应用程序也可以用来转换音频。

17.10　在网页中添加带控件的单个音频文件

让我们来实际操作一下向网页中添加音频文件。过程跟添加视频文件非常相似，只不过是使用 audio 元素，参见图 17.10.1。当然，同视频控件一样，每个浏览器都有处理音频控件外观的独特方式（参见图 17.10.2 ~ 图 17.10.6）。

```
...
<body>
    <audio src="piano.ogg" controls></audio>
</body>
</html>
```

图 17.10.1　单个 Ogg 编码的音频（包含默认控件集）。可以省略 controls 属性，但因为音频文件不可见，所以什么都不会显示



图 17.10.2　Firefox 中的音频控件。跟视频控件一样，Firefox（和 IE10）标记的音频文件要比其他浏览器标记的长 1 秒



图 17.10.3　Chrome 中的音频控件



图 17.10.4　IE10 中的音频控件



图 17.10.5　Opera 中的音频控件



图 17.10.6　Safari 中的音频控件

17

如果浏览器不支持某种音频文件格式

浏览器面对不支持的音频格式时显示的消息并不一致。例如，如果仅指定 Ogg 文件（参见图 17.10.1），IE10 会显示如图 17.10.7 所示的消息，而 Safari 则会显示播放控件和 Loading... 字样。图 17.10.4 和图 17.10.6 显示了它们支持某文件格式（如 MP3）时显示的播放控件的样子。

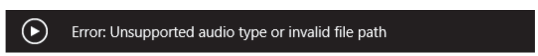


图 17.10.7　IE10 不支持你在 src 中指定的音频文件格式的情况

可用于 audio 元素的属性参见表 17.10.1。

表 17.10.1　音频属性

名　称	描　述
src（源）	指定音频文件的 URL
autoplay（自动播放）	当音频可以播放时立即开始播放
controls（控件）	添加浏览器为音频设置的默认控件
muted（静音）	让音频静音
loop（循环）	让音频循环播放
preload（预加载）	告诉浏览器要加载的音频内容的多少。可以是以下三个值： none 表示不加载任何音频；metadata 表示仅加载音频的元数据（如长度）；auto 表示让浏览器决定怎样做（这是默认的设置）

在网页中添加带控件的单个音频文件的步骤

- 获取音频文件。
- 输入 <audio src="*my-audio.ext*" controls></audio>，其中的 *my-audio.ext* 是音频文件的位置、名称和扩展名。

17.11　自动播放、循环和预加载音频

这一节介绍的属性与 video 对应的属性的原理是一样的。使用 autoplay 属性会让音频文件在页面加载时就自动播放（如图 17.11.1 和图 17.11.2 所示）。使用 loop 属性可以让音频文件循环播放（如图 17.11.3 所示）。对 preload 属性使用表 17.10.1 中的值，可以指定浏览器预加载音频文件的方式（如图 17.11.4 所示）。

```
…
<body>
    <audio src="piano.ogg" autoplay
    ➝ controls></audio>
</body>
</html>
```

图 17.11.1　当页面加载时会自动播放的 Ogg 音频文件（含默认控件集）



图 17.11.2　加载时会自动开始播放的音频文件（含控件）

```
…
<body>
    <audio src="piano.ogg" loop controls>
    ➝ </audio>
</body>
</html>
```

图 17.11.3　会循环播放的 Ogg 音频文件（含默认控件集）

```
…
<body>
    <audio src="piano.ogg" preload=
    ➝ "metadata" controls></audio>
</body>
</html>
```

图 17.11.4　页面加载时，Ogg 音频文件只加载元数据（如长度）

1. 为音频文件添加控件并让其自动播放的步骤

输入 <audio src="*my-audio.ext*" autoplay controls></audio>，这里的 *my-audio.ext* 指

向你的音频文件。如果省略 controls 属性，音频文件会自动播放，但不会显示在浏览器页面上。

2. 让音频文件循环播放的步骤

输入 <audio src="*my-audio.ext*" loop controls></audio>，这里的 *my-audio.ext* 指向你的音频文件。

3. 让浏览器仅预加载音频元数据

输入 <audio src="*my-audio.ext*" preload="metadata" controls></audio>，这里的 *my-audio.ext* 指向你的音频文件。

提 示 可以对 audio 元素包含 autoplay、loop 和 preload 属性的任意组合。注意，包含 autoplay 属性会覆盖 preload 属性的设置，因为只有加载音频文件才能播放。

提 示 17.6 节介绍的 preload 属性的 auto、none 和 metadata 值的用法说明同样适用于 audio 元素。（关于如何确定尺寸的内容并不适用于 audio。）记住，preload 属性的值不会影响浏览器的行为，它只影响请求。

17.12　提供带备用内容的多个视频源

为了获得所有兼容 HTML5 的浏览器的支持，至少需要为音频提供两种格式。实现这一目标的方法同 video 元素也是一样的，即使用 source 元素。备用方法跟 audio 元素也一样，如图 17.12.1 所示。你可能猜到了，浏览器会忽略它不支持的音频格式，而显示它支持的音频格式（参见图 17.12.2 和图 17.12.3）。不支持 audio 元素的浏览器显示备用内容，参见图 17.12.4。

```
...
<body>
    <audio controls>
        <source src="piano.ogg"
        → type="audio/ogg">
        <source src="piano.mp3"
        → type="audio/mp3">
        <p>Your browser doesn't support
        → HTML5 audio, but you can
        → <a href="piano.mp3">download the
        → audio file</a> (MP3, 1.3 MB).</
        p>
    </audio>
</body>
</html>
```

图 17.12.1　这个 audio 元素（含默认控件集）定义了两个音频源文件，一个编码为 Ogg，另一个为 MP3。完整的过程同指定多个视频源文件的过程是一样的。浏览器会忽略它不能播放的，仅播放它能播放的

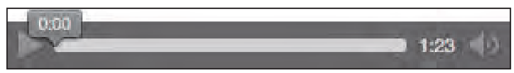


图 17.12.2　支持 Ogg 的浏览器（如 Firefox）会加载 piano.ogg。Chrome（这里并未显示）同时理解 Ogg 和 MP3，但是会加载 Ogg 文件，因为在 audio 元素的代码中，Ogg 文件位于 MP3 文件之前



图 17.12.3　不支持 Ogg 格式，但支持 MP3 格式的浏览器（IE10）会加载 piano.mp3

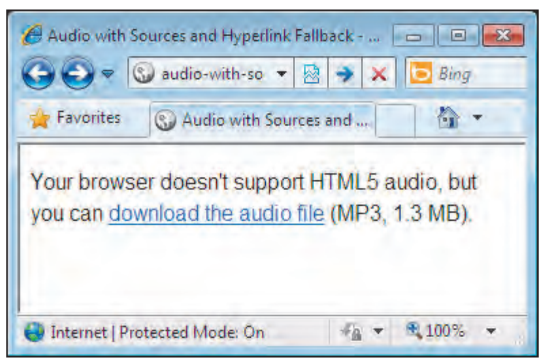


图 17.12.4　旧浏览器（如 IE8）会显示备用信息

指定两种不同的音频来源的步骤

(1) 获取音频文件。

(2) 输入 `<audio controls>` 开始 `audio` 元素（含默认控件集）。

(3) 输入 `<source src="`*my-audio.ogg*`" type="audio/ogg">`，这里的 *my-audio.ogg* 是 Ogg Vorbis 指向你的音频文件。

(4) 输入 `<source src="`*my-audio.mp3*`" type="audio/mp3">`，这里的 *my-audio.mp3* 指向你的 MP3 音频文件。

(5) 可选步骤（但推荐带着）。对于不支持 HTML5 音频元素的浏览器创建备用内容或音频下载链接。

(6) 输入 `</audio>` 结束 `audio` 元素。

> **type 属性**
>
> type 属性可以帮助浏览器判断它是否能播放某个文件。对音频文件来说，其值总是 audio/ 加上格式本身，包括 audio/ogg、audio/mp3、audio/aac、audio/wav 和 audio/mp4。

17.13　添加具有备用 Flash 的视频和音频

除了可以提供下载链接作为备用方案，还可以（或许可以说应该）嵌入一个能播放 MP4 视频文件的 Flash 备用播放器。是的，尽管 HTML5 和原生多媒体非常强大，但为了照顾那些无法处理这些技术的旧浏览器（比如 IE9 之前的版本），恐怕还得求助于嵌入 Flash 内容的方法。也就是说，如果希望获得尽可能多的用户，至少还有一种选择！

我会使用 John Dyer 的 MediaElement.js（http://mediaelementjs.com）演示如何嵌入 Flash。这并不是唯一的解决方案，但它做得不错，而且被 WordPress（一个广泛使用的博客和 CMS[①]软件）这样的系统采用了。我在提示里面还列举了一些其他的替代方案。

要让 MediaElement.js 与原生视频和音频一起运行，需要做一些额外的工作。首先，需要获取 MediaElement.js 文件，将它们放到网站文件夹里（如图 17.13.1 至图 17.13.3 所示）。然后将其中的一些文件放到网页里（如图 17.13.4 和图 17.13.5 所示）。这些步骤对视频和音频来说都是必要的。

然后，你就可以在页面中插入视频（如图 17.13.6 和图 17.13.7 所示）和音频（如图 17.13.8 和图 17.13.9 所示）了。当 MediaElement.js 由图 17.13.6 和图 17.13.8 中的脚本初始化以后，它会自动决定让浏览器使用 HTML5 原生播放系统还是 Flash 播放器。

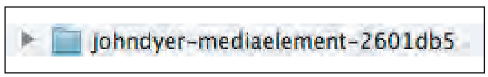


图 17.13.1　随着 MediaElement.js 的升级，ZIP 文件解压后的文件夹的名称可能与这里不一样

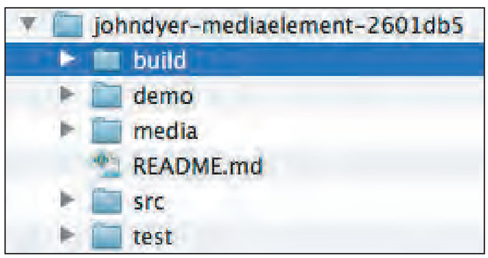


图 17.13.2　要让 MediaElement.js 运行在你的网站上，只需要 build 这一个文件夹里的内容

1. 获取 MediaElement.js 文件，并将文件包括在网站目录中

(1) 访问 http://mediaelementjs.com。点击 Download Latest 按钮，下载 ZIP 包。

① CMS 是内容管理系统的简称。——译者注

(2) 在你的电脑上找到下载下来的 ZIP 文件。该文件通常位于 Downloads 文件夹（除非你指定了别的存放路径）。解压该文件（通常是双击 ZIP 文件名），会看到一个类似于图 17.13.1 中所示的文件夹。

(3) 打开文件夹，呈现出子文件夹。复制 build 文件夹（如图 17.13.2 所示），再粘贴到网站文件夹（如图 17.13.3 所示）。

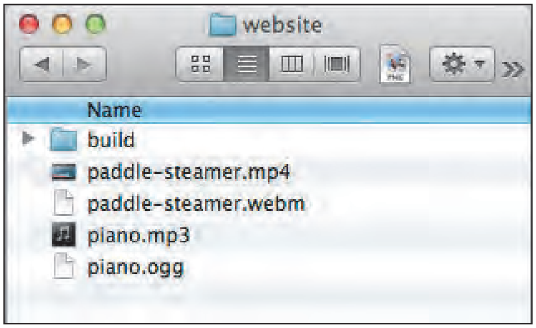


图 17.13.3　将 build 文件夹粘贴到网站的文件夹（你可能使用的是其他的文件夹名称）。为了简单，我将 build 直接放到了媒体文件所在的目录。通常，我会将媒体文件统一放到名为 media 或类似名称的文件夹里

```
<!DOCTYPE html>
<html lang="en">
<head>
     <meta charset="utf-8" />
     <title>HTML5 Media with Fallback Flash
     → Player</title>
     <script src="build/jquery.js">
      → </script>
     <script src="build/mediaelement-and-
     → player.min.js"></script>
     <link rel="stylesheet" href="build/
     → mediaelementplayer.min.css" />
</head>
<body>

</body>
</html>
```

图 17.13.4　这些文件包含了让媒体播放器得以工作的文件，还包括了统一的外观样式文件

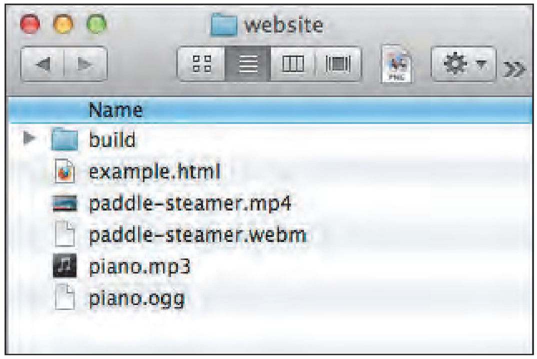


图 17.13.5　现在，有了 build 文件夹、你的网页和你的媒体文件

2. 为网页添加 MediaElement.js

页面需要加载一套特殊的文件，无论是为视频还是为音频设置备用播放器。

(1) 创建一个新的 HTML 页面，或者打开一个现有的页面。

(2) 将图 17.13.4 中突出显示的代码添加到页面中，从而让页面加载必要的样式表和 JavaScript 文件。

(3) 将页面保存到放置 build 文件夹的目录（如图 17.13.5 所示）。

3. 为视频添加备用 Flash 的步骤

(1) 获取视频资源。

(2) 输入 `<video controls>` 开始 `video` 元素（含默认控件集）。如有需要，在此处指定 `width`、`height`、`poster` 等其他属性。

(3) 输入 `<source src="*my-video.mp4*" type="type/mp4">`，这里的 *myVideo.mp4* 是 MP4 视频源文件的名称。

(4) 输入 `<source src="*myVideo.webm*"type="video/webm">`，这里的 *my-video.webm* 是 WebM 视频源文件的名称。

(5) 如有需要，为不支持 HTML5 视频和 Flash 的浏览器提供备用信息和链接。

(6) 输入 </video> 结束 video 元素。

(7) 将图 17.13.6 中突出显示的代码添加到页面中，从而初始化一个视频播放器。将 script 放到 </body> 的前面，哪怕页面拥有视频以外的其他内容。

现在，你的视频拥有了尽可能广泛的受众，参见图 17.13.7。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>HTML5 Media with Fallback Flash
    → Player</title>
    <script src="build/jquery.js"></script>
    <script src="build/mediaelement-and-
    → player.min.js"></script>
    <link rel="stylesheet" href="build/
    → mediaelementplayer.min.css" />
</head>
<body>
<video width="369" height="208"
→ preload="metadata" controls>
    <source src="paddle-steamer.mp4"
    → type="video/mp4">
    <source src="paddle-steamer.webm"
    → type="video/webm">
    <p>Your browser doesn't support HTML5
    → video or Flash, but you can <a href=
    → "paddle-steamer.mp4">download the
    → video</a> (MP4, 2.4 MB).</p>
</video>

<!-- 以下代码要放在所有内容下面 -->
<script>
$('video').mediaelementplayer();
</script>
</body>
</html>
```

图 17.13.6　这段视频代码看起来很熟悉。事实上，它与先前引用多个来源的代码是一样的。根据需要，可以添加其他的属性或删除一部分属性。底部的脚本告诉 MediaElement.js 施展它的魔法



图 17.13.7　无论使用哪种浏览器，无论是否针对不支持 HTML5 视频的浏览器使用 Flash 版本，视频的播放控件看起来都是一样的

```
... 页面顶端跟图17.13.4和图17.13.6一样 ...

<body>
<audio controls>
    <source src="piano.ogg" type="audio/ogg">
    <source src="piano.mp3" type="audio/mp3">
    <p>Your browser doesn't support HTML5
    → audio or Flash, but you can <a href=
    → "piano.mp3">download the audio file
    → </a> (MP3, 1.3 MB).</p>
</audio>

<!-- 下面的代码放在所有内容之后 -->
<script>
$('audio').mediaelementplayer();
</script>
</body>
</html>
```

图 17.13.8　这段代码与图 17.13.6 中代码的唯一区别就是这里指定的是 audio 而非 video。audio 元素看起来也相当熟悉



图 17.13.9　同视频一样，无论在哪个浏览器，无论是否使用 Flash 版本，音频的播放控件看起来都是一样的

```
...

<script>
$('audio,video').mediaelementplayer();
</script>
</body>
</html>
```

图 17.13.10 以下代码会初始化页面中所有的音频和视频

4. 为音频添加备用 Flash 的步骤

(1) 获取音频文件。

(2) 输入 `<audio controls>` 开始 audio 元素（含默认控件集）。

(3) 输入 `<source src="my-audio.ogg"type="audio/ogg">`，这里的 *my-audio.ogg* 指向你的 Ogg Vorbis 音频文件。

(4) 输入 `<source src="my-audio.mp3"type="audio/mp3">`，这里的 *my-audio.mp3* 指向你的 MP3 音频文件。

(5) 如果需要，为不支持 HTML5 音频或者 Flash 的旧浏览器创建信息和链接。

(6) 输入 `</audio>` 结束 audio 元素。

(7) 将图 17.13.8 中突出显示的代码添加到页面中去，从而初始化音频播放器。将 script 放到 `</body>` 的前面，哪怕页面拥有音频以外的其他内容。

现在，你的音频拥有了尽可能广泛的受众，参见图 17.13.9。一天便有了两个成就！

提示 如果页面同时包含了视频和音频，那么需要修改页面询问的脚本，如图 17.13.10 所示。

提示 Internet Explorer 8 等浏览器会忽略 audio 和 source 元素，直接使用备用 Flash 播放器。只要用户安装了 Flash，就能播放视频或音频内容。

提示 如果没有为视频指定 preload="metadata"（就像图 17.13.6 所示的那样），也没有指定海报图像，那么视频的 Flash 版本会显示一个黑色的矩形而不是视频的一帧。不过，当视频开始播放时，黑色的矩形就会被视频本身替代。

提示 如果在 video 开始标签中指定了 width 和 height，那么视频的 Flash 版本就有可能显示黑边。不过，如果不指定尺寸，视频会在一开始呈现一个比正常尺寸要大的区域，然后再缩小到正常的大小。

提示 其他的 Flash 播放解决方案包括 Video.js（www.videojs.com）、JW Player（www.longtailvideo.com/jw-player/）、Flowplayer（http://flowplayer.org） 等。JW Player 和 Flowplayer 的免费版本会在媒体播放器上显示它们的标识。

17

当 Flash 播放器不起作用时

由于 Flash 的安全设置，当你在自己的电脑上测试 Flash 播放器（即所有的文件都位于自己的电脑上，而不是在服务器上）时，有可能无法播放媒体文件。

一种解决办法是将你的 MP3 和 MP4 文件上传到 Web 服务器，并在 HTML 中使用绝对路径引用它们。

例如，将图 17.13.6 中的文件上传到服务器的 media 文件夹中。可以将第一个 src 值修改如下（将 www.yourdomain.com 替换为真实域名）：

```
<source src="http://www.yourdomain.com/
→ media/paddle-steamer.mp4" type="video/
→ mp4">
```

> 然后保存 HTML 页面，再在你的电脑上测试（HTML 页面不必位于服务器上）。你还可以修改 Flash 安全设置，允许本地目录播放 Flash。参见 http://mediaelementjs.com/#installation。
>
> 如果还是有问题，参见 17.2 节的补充材料“设置 MIME 类型”。

17.14　高级多媒体

使用 HTML5 原生多媒体的另一个好处是可以利用很多来自 HTML5 或与 HTML5 相关的新特性和新功能。本节主要讨论其中的两个：canvas 元素和 SVG。

1. 通过 canvas 操作视频

使用 canvas 元素及相应的 JavaScript API 可以在网页上描制并创建动画。

可以对 HTML5 视频应用这些 API，因为 video 元素可以同其他 HTML 元素一样进行处理，因此它也可以被 canvas 访问和获取。

通过 JavaScript API，可以从播放的视频中抓取图像，并在 canvas 元素中重新绘制该图像，从而创建视频的截图。

通过 API 可以对单个图像像素进行操作，同时由于可以根据视频在 canvas 中创建图像，因而可以调整视频的像素。例如，可以将视频转化为灰度模式。

这只是让你对通过 canvas 操作 video 元素建立一些简单的概念，对这一主题的深入探讨已经超出了本书的范围。

2. 联合使用 SVG 和视频

人们开始关注的另一项与 HTML5 有关的技术是 SVG（Scalable Vector Graphic，可缩放矢量图形）。

SVG 已经存在相当一段时间了（它诞生自 1999 年），但直到 HTML5 才有了 svg 元素。通过该元素可以在网页本身嵌入 SVG 定义。

SVG 使用 XML 定义图形和图像，浏览器则对其进行解释和使用，从而描绘出真正的图形。SVG 定义所包含的全部内容就是对如何绘制和绘制什么的说明。

使用 SVG 创建的图像也是基于矢量而不是基于光栅的。这意味着它们可以很好地适应缩放，因为浏览器只是简单地依照绘制说明，根据所需的尺寸，将图形绘制出来。相较而言，GIF、PNG 和 JPEG 文件等光栅光栅图像包含的是像素数据，如果要以远大于原始图像的尺寸重新绘制图像，就会因为缺少足够的像素数据导致图像质量受损。

关于 SVG 的完整讨论超出了本章的范围，这里只是想让你知道视频可以同 SVG 定义联合使用。通过 SVG 创建的图形可以用于对视频进行遮罩，也即只显示能透过该图形（如圆圈）的底层视频。还可以创建可调整为任意大小的自定义视频控件。

此外，还有一些 SVG 滤镜可以应用于 HTML5 视频，如黑白转换、高斯模糊、色彩饱和度等。

17.15　更多资源

本章仅涵盖了 HTML5 多媒体的基础知识。关于这一主题，还有很多值得学习的东西。这里有大量有关资源供你研究。

1. 在线资源

- “Video on the Web”（http://diveinto.html5doctor.com/video.html）
- “WebVTT and Video Subtitles”（www.iandevlin.com/blog/2011/05/html5/webvtt-and-video-subtitles）
- “HTML5 Canvas: The Basics”（http://dev.opera.com/articles/view/html-5-canvas-the-basics）
- “Learning SVG”（http://my.opera.com/tagawa/blog/learning-svg）

2. 图书

- Ian Devlin 所著的 *HTML5 Multimedia: Develop and Design*（Peachpit Press，2011），参见 http://html5multimedia.com
- Shelley Powers 所著的 *HTML5 Media*（O’Reilly Media，2011）
- Silvia Pfeiffer 所著的 *The Definitive Guide to HTML5 Video*（Apress，2010）

第 18 章

表格

18

本章内容

- 结构化表格
- 让单元格跨越多列或多行

在日常生活中，我们对表格式数据已经很熟悉了。这种数据有多种形式，如财务数据、调查数据、事件日历、公交车时刻表、电视节目表等。在大多数情况下，这类信息都由列标题或行标题加上数据本身构成。

本章将对 table 元素及其子元素进行讲解，重点是基本的 table 结构和样式。HTML 表格可以很复杂，不过很少需要实现特别复杂的表格（除非是具有丰富数据的网站）。对于一些比较复杂的示例，参见以下 URL。

- Roger Johansson 的“Bring On the Tables”（www.456bereastreet.com/archive/200410/bring_on_the_tables/）
- Roger Hudson 的“Accessible Data Tables”（www.usability.com.au/resources/tables.cfm）
- Stephen Ferg 的“Techniques for Accessible HTML Tables”（http://accessiblehtml.sourceforge.net/）

18.1 结构化表格

放在电子表格中的信息通常很适合用 HTML 表格呈现。

从基本层面看，table 元素是由行组成的，行又是由单元格组成的。每个行（tr）都包含标题单元格（th）或数据单元格（td），或者同时包含这两种单元格。如果认为为整个表格添加一个标题有助于访问者理解该表格，可以提供 caption。在浏览器中，标题通常显示在表格上方，其用途显而易见。此外，scope 属性（也是可选的，不过推荐使用）可以告诉屏幕阅读器和其他辅助设备当前的 th 是列的标题单元格（使用 scope="col"）还是行的标题单元格（使用 scope="row"），抑或是用于其他目的的单元格（参见最后一条提示）。表格元素的基本实现代码如图 18.1.1 所示。

在默认情况下，表格在浏览器中呈现的宽度是其中的信息在页面可用空间里所需要的最小宽度，如图 18.1.2 所示。很容易就会想到可以通过 CSS 改变表格的格式，后面很快就会讲到。

不过，图 18.1.1 中的表格似乎缺少一些东西。如何知道每行数据表示的是什么呢？如果每个行也有标题单元格，就很容易理解了。添加这些单元格只需要在每行开头添加一个 th 元素就可以了。列标题应设置 scope="col"，而每个行的 th（位于 td 之前）则应设置 scope="row"，如图 18.1.3 所示。

```
...
<body>

<table>
    <caption>Quarterly Financials for
    → 1962-1964 (in Thousands)</caption>
    <tr>
        <th scope="col">1962</th>
        <th scope="col">1963</th>
        <th scope="col">1964</th>
    </tr>
    <tr>
        <td>$145</td>
        <td>$167</td>
        <td>$161</td>
    </tr>
    <tr>
        <td>$140</td>
        <td>$159</td>
        <td>$164</td>
    </tr>
    <tr>
        <td>$153</td>
        <td>$162</td>
        <td>$168</td>
    </tr>
    <tr>
        <td>$157</td>
        <td>$160</td>
        <td>$171</td>
    </tr>
</table>

</body>
</html>
```

图 18.1.1 每个行都是由 tr 元素标记的。这是个很简单的表格，它只有一个包含标题单元格（th 元素）的行和三个包含数据单元格（td 元素）的行。本例中也包含了 caption 元素，不过它是可选的，参见第一条提示

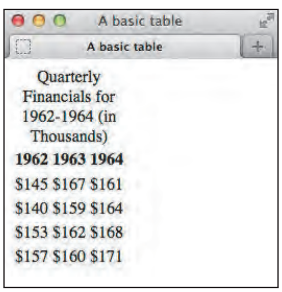


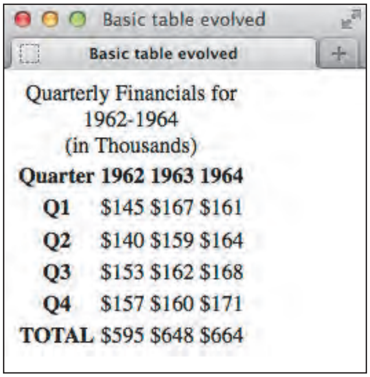
图 18.1.2 在默认情况下，th 文本是以粗体显示的，th 和 caption 文本都是居中对齐的，表格的宽度就是内容所需的宽度

```
...
<body>

<table>
    <caption>Quarterly Financials for
    → 1962-1964 (in Thousands)</caption>
    <thead> <!-- 表格头部 -->
        <tr>
            <th scope="col">Quarter</th>
            <th scope="col">1962</th>
            <th scope="col">1963</th>
            <th scope="col">1964</th>
        </tr>
    </thead>
    <tbody> <!-- 表格主体 -->
        <tr>
            <th scope="row">Q1</th>
            <td>$145</td>
            <td>$167</td>
            <td>$161</td>
        </tr>
        <tr>
            <th scope="row">Q2</th>
            <td>$140</td>
            <td>$159</td>
            <td>$164</td>
        </tr>
        ... [Q3 and Q4 rows] ...
    </tbody>
    <tfoot> <!-- 表格尾部 -->
        <tr>
            <th scope="row">TOTAL</th>
            <td>$595</td>
            <td>$648</td>
            <td>$664</td>
        </tr>
    </tfoot>
</table>

</body>
</html>
```

图 18.1.3 通过指定 thead、tbody 和 tfoot 显式地定义了表格的不同部分。接着，在每行的开头添加了 th 元素。tbody 和 tfoot 中的 th 设置了 scope="row"，表明它们是行标题。表格在浏览器中的显示效果如图 18.1.4 所示



Quarter	1962	1963	1964
Q1	$145	$167	$161
Q2	$140	$159	$164
Q3	$153	$162	$168
Q4	$157	$160	$171
TOTAL	$595	$648	$664

图 18.1.4　这个表既有列标题，也有行标题。由于添加了新列，表格变宽了，因此整个表格的标题也比图 18.1.2 中的更宽一些

图 18.1.3 中还特意引入了其他一些专门用于定义表格的元素，即 thead、tbody 和 tfoot。thead 元素可以显式地将一行或多行标题标记为表格的头部。tbody 元素用于包围所有的数据行。tfoot 元素可以显式地将一行或多行标记为表格的尾部。可以使用 tfoot 包围对列的计算值（就像图 18.1.3 中所示的那样），也可以在长表格（如列车时刻表）中使用 tfoot 重复 thead 中的内容（如果表格在打印时超过一页，有的浏览器会在每一页都打印 tfoot 和 thead 元素的内容）。thead、tfoot 和 tbody 元素不会影响表格的布局，也不是必需的（不过推荐使用它们）。如果包含了 thead 或 tfoot，则必须同时包含 tbody。此外，还可以对它们添加样式。

创建表格结构的步骤

(1) 输入 <table>。

(2) 如果需要，输入 <caption>*caption content*</caption>，其中的 *caption content* 是对表格的描述。

(3) 如果需要，在要创建的表格部分的第一个 tr 元素之前，输入 <thead>、<tbody> 或 <tfoot>（需要说明的是，<tbody> 不能在 <thead> 之前）。

(4) 输入 <tr> 定义行的开始。

(5) 输入 <th scope="*scopetype*"> 开始标题单元格（其中的 *scopetype* 可以是 col、row、colgroup 或 rowgroup），或者输入 <td> 定义数据单元格的开始。

(6) 输入单元格的内容。

(7) 输入 </th> 结束标题单元格，或者输入 </td> 结束数据单元格。

(8) 对行内的每个单元格重复第 (5) 步至第 (7) 步。

(9) 输入 </tr> 结束行。

(10) 为所在表格部分内的每个行重复第 (4) 步至第 (9) 步。

(11) 如果在第 (3) 步中开始了一个表格部分，就根据需要使用 </thead>、</tbody> 或 </tfoot> 结束这个部分。

(12) 为每个表格部分重复第 (3) 步至第 (11) 步。注意，一个表格只能有一个 thead 和 tfoot，但可以有多个 tbody 元素。

(13) 输入 </table> 以结束表格。

如图 18.1.4 所示，在默认情况下，表格被压得很扁。通过应用一些 CSS（如图 18.1.5 所示），可以为单元格添加一些空间让它们扩大一些（使用 padding），添加边框以指示单元格的边界（使用 border），还可以对文本进行格式化。这些样式都有助于更好地理解表格，如图 18.1.6 所示。

提 示　如果包含了 caption 元素，那么它必须是 table 中的第一个元素，如图 18.1.1 所示。（caption 元素可以包含 p 和其他文本元素。）

提 示　如果包含了 thead 或 tfoot，则必须包含 tbody。tbody 不能位于 thead 之前。一个 table 只能拥有一个 thead 和一个 tfoot，但可以有多个 tbody 元素。

```
body {
    font: 100% "Courier New", Courier,
    → monospace;
}

table {
    border-collapse: collapse;
}

caption {
    font-size: .8125em;
    font-weight: bold;
    margin-bottom: .5em;
}

th,
td {
    font-size: .875em;
    padding: .5em .75em;
}

td {
    border: 1px solid #000;
}

tfoot {
    font-style: italic;
    font-weight: bold;
}
```

图 18.1.5　这个简单的样式表为每个数据单元格添加了 border，为标题单元格和数据单元格都添加了 padding，还对表格 caption 和内容进行了格式化。如果不对 table 定义 border-collapse: collapse;，每个 td 的边框和相邻 td 的边框之间就会出现一些空隙（默认值为 border-collapse: separate;）。还可以像 18.2 节所示的那样，对 th 元素应用边框

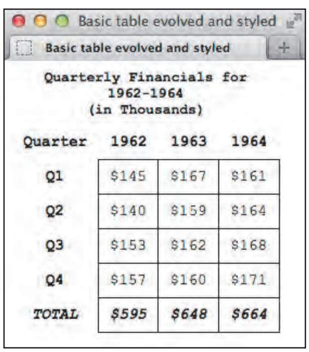


图 18.1.6　现在，表格既有列标题，也有行标题，还有一个显示各列数据之和的行（位于 tfoot 元素中）。HTML 还没有聪明到为你计算这些东西的数量，因此确保行列数都正确

提示　如果 table 是嵌套在 figure 元素内除 figcaption 以外的唯一元素，则可以省略 caption，使用 figcaption 对表格进行描述（参见 4.4 节）。注意，不要在 table 中嵌套 figcaption，而应跟往常一样将 figcaption 放在 figure 中。

提示　还可以在样式表中对 table、td 或 th 元素设置 background、width 等属性，尽管没有在 CSS 示例中进行演示，如图 18.1.4 所示。总之，大多数用于其他 HTML 元素的文本格式和其他格式也可以应用于表格（参见 18.2 节的另一个例子）。你可能会发现，不同的浏览器显示的样式稍有差异，尤其是 Internet Explorer。

提示　可以通过 scope 属性指定 th 为一组列的标题（使用 scope="colgroup"），或者为一组行的标题（使用 scope="rowgroup"）。对于后者，参见下一节的示例。

18.2　让单元格跨越多列或多行

可以通过 colspan 和 rowspan 属性让 th 或 td 跨越一个以上的列或行。对该属性指定的数值表示的是跨越的单元格的数量，如图 18.2.1 和图 18.2.2 所示。

1. 让单元格跨越两个或两个以上列的步骤

(1) 在需要定义跨越一个以上的列的单元格的地方，如果为标题单元格，输入 <th 后加一个空格，否则输入 <td 后加一个空格。

(2) 输入 colspan="*n*">，这里的 *n* 是单元格要跨越的列数。

(3) 输入单元格的内容。

(4) 根据前面的内容，输入 </th> 或者 </td>。

(5) 根据 18.1 节中的介绍，完成表格的其余部分。如果创建了一个跨越两列的单元格，在该行就应该少定义一个单元格；如果创建了一个跨越三列的单元格，在该行就应该少定义两个单元格，以此类推。

```
...
<body>

<table>
    <caption>TV Schedule</caption>
    <thead> <!-- 表格头部 -->
      <tr>
        <th scope="rowgroup">Time</th>
        <th scope="col">Mon</th>
        <th scope="col">Tue</th>
        <th scope="col">Wed</th>
      </tr>
    </thead>
    <tbody> <!-- 表格主体 -->
      <tr>
        <th scope="row">8 pm</th>
        <td>Staring Contest</td>
        <td colspan="2">Celebrity
        → Hoedown</td>
      </tr>
      <tr>
        <th scope="row">9 pm</th>
        <td>Hardy, Har, Har</td>
        <td>What's for Lunch?</td>
        <td rowspan="2">Screamfest Movie
        → of the Weak</td>
      </tr>
      <tr>
        <th scope="row">10 pm</th>
        <td>Healers, Wheelers &amp;
        → Dealers</td>
        <td>It's a Crime</td>
      </tr>
    </tbody>
</table>

</body>
</html>
```

图 18.2.1　通过在包含 *Celebrity Hoedown* 演出的 td 上应用 colspan="2"，指示该演出的上映时间包含周二和周三晚上 8 点这两个时间。类似地，在包含 *Screamfest Movie of the Weak* 的 td 上添加了 rowspan="2"，因为该演出持续两个小时。同时，注意 Time 这个 th 设置了 scope="rowgroup"，因为它是它正下方的每个行标题组的标题

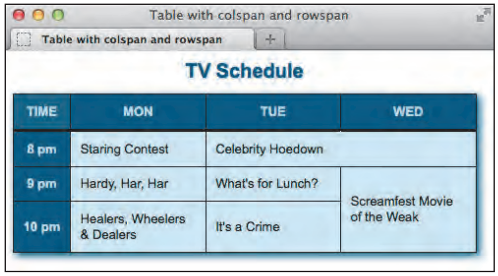


图 18.2.2　仅仅通过查看代码，很难判断 colspan 和 rowspan 对表格的影响，但在浏览器中查看就清楚多了。这个表格还用 CSS 添加了一些样式，完整的样式表见本书的配套网站 www.htmlcssvqs.com/8ed/18

2. 让单元格跨越两个或两个以上行的步骤

(1) 在需要定义跨越一个以上的行的单元格的地方，如果为标题单元格，输入 <th 后加一个空格，否则输入 <td 后加一个空格。

(2) 输入 rowspan="*n*">，这里的 *n* 是单元格要跨越的行数。

(3) 输入单元格的内容。

(4) 根据前面的内容，输入 </th> 或者 </td>。

(5) 根据 18.1 节中的介绍，完成表格的其余部分。如果创建了一个 rowspan 等于 2 的单元格，就不需要定义下一行中该单元格对应的单元格了；如果创建了一个 rowspan 等于 3 的单元格，就不需要定义下面两行中该单元格对应的单元格了，以此类推。

提示　表格中的每一行都应具有相同的单元格数量。跨越多列的单元格应算做多个单元格，它的 colspan 属性值为多少，就算做多少个单元格。

提示　表格中的每一列都应具有相同的单元格数量。跨越多行的单元格应算做多个单元格，它的 rowspan 属性值为多少，就算做多少个单元格。

第 19 章 添加 JavaScript

19

本章内容

- 加载外部脚本
- 添加嵌入脚本
- JavaScript 事件

HTML 定义网页的内容，CSS 定义网页的表现，JavaScript 则定义特殊的行为。建立网站不可能脱离 HTML（如果要让网站看起来很吸引人，则离不开 CSS），但 JavaScript 并不是必需的。在大多数情况下，JavaScript 的特性都是用于增强访问者体验的——它们在由 HTML 和 CSS 构建的核心体验的基础上进行增强。（参见本书前言部分“渐进增强：一种最佳实践”）。

通过编写简单的 JavaScript 程序，可以显示和隐藏内容；通过编写复杂一些的程序，可以加载数据并动态地更新页面。可以操作定制的 HTML5 `audio` 和 `video` 元素控件，使用 HTML5 的 `canvas` 元素创建游戏。可以利用地理定位，根据访问者所在的位置定制其体验；可以让访问者通过将文件拖放到浏览器窗口进行上传（Dropbox 的网站就是一个这样做的例子）。可以利用 JavaScript 和一些强大的 HTML5 特性及相关技术构建成熟的 Web 应用程序（这些内容都是高级特性，因此不在本书讲解范围之内）。

如你所见，JavaScript 的功能非常强大，而它的使用也呈现了爆炸式的增长。jQuery（jquery.com）等 JavaScript 库确保了为页面添加简单交互和复杂行为的过程变得容易了许多。还有很多跟 jQuery 功能类似的库，但在这些库中，jQuery 是用得最多的一个，这主要是因为初学者很容易上手，同时它有很好的在线文档和大型社区支持。除了 jQuery 以外，还有各式各样的 JavaScript 可用于帮你构建大型 Web 应用。使用 Node.js（http://nodejs.org）甚至可以用 JavaScript 构建 Web 服务器。

浏览器厂商花费了大量的精力确保其浏览器处理 JavaScript 的速度较几年前的版本有了显著提升。JavaScript 也可以在平板电脑和现代移动浏览器中运行，不过出于性能方面的原因，可能需要为这些设备考虑在页面中加载的脚本的大小。

不过 JavaScript 本身是一个独立、庞大的主题，因此我们不会在本书讲解这门语言。我仍会解释如何将创建好的脚本插入到 HTML 文档中去，同时给出一些关于如何在插入脚本时尽量降低其对页面影响的建议，此外还会提供对事件处理程序的概览。

当你熟悉 HTML 和 CSS 后，我建议你开始学习 JavaScript（以及 jQuery）。Marijn Haverbeke 的 *Eloquent JavaScript*（中文版《JavaScript 编程精解》，机械工业出版社，

2012）是开始学习 JavaScript 的好资料。该书的原始版本是免费的，位于 http://eloquentjavascript.net，此外还有修订后的纸质版。当你对 JavaScript 语言有一定了解后，可以看看 Ivo Wetzel 和 Zhang Yi Jiang 的 JavaScript Garden（http://bonsaiden.github.io/JavaScript-Garden/），这是个学习 JavaScript 语言精华和怪异特性的免费、简洁的资源。

JavaScript 要比 HTML 和 CSS 更复杂一些，因此，如果你发现自己要花费较长的时间学习，也不要感到气馁。对任何事物来说，你操作越多，就越熟练。

19.1 加载外部脚本

脚本主要有两种类型，一种是从外部文件（使用纯文本格式）加载的脚本，另一种是嵌入在页面中的脚本（这一种将在下一节讲解）。这同外部样式表和嵌入样式表的概念是类似的。

同为页面添加样式表一样，从外部文件加载脚本通常比在 HTML 中嵌入脚本要好一些（如图 19.1.1 所示）。这样做的好处也是类似的，即可以在需要某一脚本的每个页面加载同一个 JavaScript 文件。需要对脚本进行修改时，就可以仅编辑一个脚本，而不是在各个单独的 HTML 页面更新相似的脚本。

无论是加载外部脚本还是嵌入脚本，均使用 script（脚本）元素。

加载外部脚本的方法

输入 `<script src="script.js"></script>`，这里的 *script.js* 是外部脚本在服务器上的位置及文件名。应该尽可能地将脚本元素放在 `</body>` 结束标签之前（参见图 19.1.1），而不是放在文档的 head 元素里（参见图 19.1.2）。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Loading an External Script</title>
    <link rel="stylesheet"
    ➝ href="css/global.css" />
</head>
<body>
... 所有的HTML内容写在这里 ...

<script src="behavior.js"></script>
</body>
</html>
```

图 19.1.1 script 元素的 src 属性引用脚本的 URL。在大多数情况下，最好在页面的最末尾加载脚本，即 `</body>` 结束标签的前面。也可以在页面的 head 元素中加载脚本，如图 19.1.2 所示，但这样做会影响页面显示的速率。更多信息参见补充材料“脚本和性能的最佳实践”

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Loading an External Script</title>
    <!-- 在加载任何JS文件之前加载样式表 -->
    <link rel="stylesheet" href="css/
    ➝ global.css" />
    <script src="behavior.js"></script>
</head>
<body>
... 所有的HTML内容写在这里 ...
</body>
</html>
```

图 19.1.2 在这个例子中，脚本是在 head 中加载的。它位于 link 元素的后面，因此不会影响 CSS 文件先于脚本开始加载。关于为什么要尽可能地避免在 head 中加载脚本，参见“脚本和性能的最佳实践”

提示　为了保持组织文件良好，通常将 JavaScript 文件放在一个子文件夹中（常用的名称包括 js、scripts 等），参见 2.6 节。src 属性需要反映这一点，就像指向某一资源的其他 URL 一样。例如，如果图 19.1.1 中的文件位于名为 assets/js/ 的文件夹内，则应输入 `<script src="assets/js/behavior.js"></script>`。（这只是一个示例，还有其他表示 URL 的方式，参见 1.7 节。）

提示　图 19.1.3 显示了一段 JavaScript 示例。由于 JavaScript 是纯文本，因此你可以在创建 HTML 和 CSS 的同一编辑器中编写 JavaScript。如果这个例子保存为 behavior.js，它就会在图 19.1.1 和图 19.1.2 所示的页面中加载。

```
/*
  当用户选择 Other 单选按钮时，textarea 会变为可用的，光标也会显示在其中，用户可以马上输入。选择其余
  两个单选按钮中的任意一个，则会禁用 textarea
 */

(function (window, document) {
     'use strict';

     var choices = document.getElementById('choices'),
        textarea = document.getElementById('other-description');

     if (!choices || !textarea) {
        return;
     }

     // 默认禁用textarea
     textarea.disabled = true;

     // 为单选按钮添加行为
     choices.onclick = function(e) {
        var target,
           e;

        if (!e) {
           e = window.event;
        }

        target = e.target || e.srcElement;

        // 根据所选的单选按钮，切换textarea的状态
        if (target.getAttribute('type') === 'radio') {
           if (target.id !== 'other') {
              textarea.disabled = true;
           } else {
              textarea.disabled = false;
              textarea.focus();
           }
        }
     };
}(window, document));
```

图 19.1.3　如你所见，JavaScript 与 HTML 和 CSS 相当不一样！这段脚本与 16.16 节引用的 toggle-textarea.js 几乎是一样的。本书配套网站上的例子包含了额外的注释，解释了代码的工作原理（参见 www.htmlcssvqs.com/8ed/19）。即便如此，我也不会具体地讲解代码的方方面面，如果你刚开始接触 JavaScript，你会发现不懂的地方很多。我提供这个例子，主要是为了让你对这门语言有个大致的感受

提示 一个页面可以加载多个 JavaScript 文件，可以包含多个嵌入的脚本（参见图 19.2.1）。在默认情况下，浏览器会按照脚本在 HTML 中出现的顺序对它们进行加载（如果需要的话）和执行。关于为什么需要尽可能地避免加载多个脚本以及如何简化脚本（参见图 19.1.4），参见补充材料“脚本和性能的最佳实践”。

```
(function(e,c){var d=c.getElementById
→ ("choices"),b=c.getElementById("other-
→ description");d&&b&&(b.disabled=!0,d.
→ onclick=function(a){a||(a=e.event);a=
→ a.target||a.srcElement;"radio"===
→ a.getAttribute("type")&&("other"!==
→ a.id?b.disabled=!0:(b.disabled=!1,
→ b.focus()))})})(window,document);
```

图 19.1.4　是的，这段代码与图 19.1.3 的代码是一样的，只是被压缩过了。它看起来就像是一只猫在键盘上跑过留下的字符，但实际上浏览器可以理解它们

提示 可以为外部脚本指定任意有效的文件名，只要它们以 .js 结尾。通常，压缩后的脚本以 .min.js 作为扩展名，这样可以很清晰地将它们与常规文件区分开来。两个版本的文件都要保留——在常规文件中更新代码（因为代码更易于阅读），但在网站上使用的则是压缩后的版本（因为对浏览器来说能更快地解析）。不要忘了每次更新完常规文件后都要生成一个新的压缩文件，否则，访问者看到的仍是旧的版本。

提示 如果创建了一个简化文件，一定要确保在 HTML 中对脚本的引用是正确的，例如，`<script src="behavior.min.js">`。否则，页面仍会加载常规的文件，访问者不会获得更小的文件带来的任何好处。在不断修改脚本的过程中，也可以让 `src` 重新指向常规的文件。

提示 不理解 JavaScript 的浏览器（数量很少）或用户禁用了 JavaScript 的浏览器会忽略 JavaScript 文件。因此要确保页面不依赖于 JavaScript 为用户提供对内容的可访问性和基本体验。（高度依赖 JavaScript 的 Web 应用程序通常例外。）

提示 从技术上说，为页面添加 JavaScript 还有第三种方式：内联脚本。**内联脚本**是直接在 HTML 中特定元素属性上指定的 JavaScript 片段。在这里提到这种方式是为了告诉你，应该避免使用它们，就像避免使用内联样式表一样。正如内联样式表将 HTML 和 CSS 混合在一起，内联脚本则将 HTML 和 JavaScript 纠缠在一起，这与将它们分开的最佳实践不符。

脚本和性能的最佳实践

关于脚本和页面性能最佳实践的详细讨论超出了本书的范围，不过这里仍会提到几个影响较大的要点。

最重要的一点，理解浏览器如何处理脚本是很有用的。随着页面的加载，在默认情况下，浏览器会按照脚本在 HTML 中出现的顺序依次对每个脚本进行下载（对于外部脚本）、解析和执行。在处理脚本的过程中，浏览器既不会下载该 `script` 元素后面出现的内容（哪怕是文本），也不会呈现这些内容。这称为**阻塞行为**（blocking behavior）。

这条规则对嵌入脚本和外部脚本都有效。可以想象，这会影响页面的呈现速率，影响的程度取决于脚本的大小和它执行的动作。

大多数浏览器都会这样做，因为 JavaScript 可能包含其他脚本所依赖的代码、立即生成内容的代码或对页面进行调整的代码。浏览器需要在结束呈现页面之前考虑所有这些问题。

那么如何避免这一点呢？消除 JavaScript 阻塞最简单的方法就是将所有的 `script` 元素放置在 HTML 结束之前，即 `</body>` 结束标签的前面，参见图 19.1.1。

随便花点儿时间浏览一下其他人的网站，不难发现脚本是在 head 元素中加载的。除了少数必须这样做的情况，通常认为这是一种过时的做法，应当尽可能地避免。（必须这样做的一种情况就是加载第 11 章所述的 HTML5 剃刀。）如果确实需要在 head 中加载脚本，也要将它们放置在所有加载 CSS 文件的 `link` 元素之后（这也是出于性能的考虑）。

另一种简单的加快脚本加载速率的方法就是将 JavaScript 放在同一个文件中（或尽可能少的一些文件中）并**压缩**代码。通常，压缩代码会去除换行、注释及额外的空格（以及其他同未压缩的代码存在差异的地方）。可以想象一下只在一个很长的行内编写代码，而永远不敲回车键。看看图 19.1.4 你就明白我说的是什么意思了。

可以使用以下工具压缩脚本（“供下载的版本及文档”链接主要是为高级用法提供的）：

- Google Closure Compiler
  http://code.google.com/closure/compiler/（供下载的版本及文档）
  http://closure-compiler.appspot.com（在线版本）
- UglifyJS（使用第二个链接）
  https://github.com/mishoo/UglifyJS2（供下载的版本及文档）
  http://lisperator.net/uglifyjs/（在线版本，选择“Open demo”）
- YUI Compressor（使用第二个链接）
  http://developer.yahoo.com/yui/compressor/（供下载的版本及文档）
  http://refresh-sf.com/yui/（非官方的在线版本）

上述每个工具都能减小文件的大小，而压缩的结果则因脚本而异。记住，有时压缩代码的工具效果过于强烈，可能会损坏你的脚本。因此，如果页面包括压缩脚本，一定要进行测试。通常，浏览器加载一个文件比加载两个文件（或更多的文件）要快一些，即便单个文件的大小比不同文件的总大小还要大一些（除非大得多）。

这是两种常用且强大的方法，但真正说起来，它们也只是一些皮毛的东西。关于脚本加载的方法及其优化，强烈推荐 Steve Souders 所著的 *Even Faster Web Sites*[①]（O’Reilly Media，2009）一书，以及他的网站 www.stevesouders.com。不过有言在先，Souders 的某些讨论相对技术一些。

19

① 中文版《高性能网站建设进阶指南》已由电子工业出版社出版。——译者注

19.2　添加嵌入脚本

嵌入脚本位于 HTML 文档之内，同嵌入样式表很相似。嵌入的脚本包含在 script 元素内，如图 19.2.1 所示。嵌入脚本并不是首选的方法（参见 19.1 节），但有时必须这样做。

```
<!DOCTYPE html>
<html lang="en">
<head>
     <meta charset="utf-8" />
     <title>Adding an Embedded Script</title>
     <link rel="stylesheet"
     → href="css/global.css" />
</head>
<body>
... 所有HTML内容写在这里 ...

<script>
/*
JavaScript 代码写在这里

*/
</script>
</body>
</html>
```

图 19.2.1　嵌入脚本没有 src 属性。相反，代码位于页面之内。如果要嵌入脚本，应该尽可能地在 </body> 结束标签之前这样做。虽然也可以在 head 中嵌入脚本（如图 19.2.2 所示），但通常出于性能原因而避免这样做

添加嵌入脚本的步骤

(1) 在 HTML 文档中，输入 <script>。

(2) 输入脚本的内容。

(3) 输入 </script>。

提示　每个脚本都按照它们出现在 HTML 中的顺序依次进行处理，无论是嵌入脚本还是外部脚本（参见 19.1 节）。

提示　尽管 script 元素必须有结束标签（</script>），但在有 src 属性的情况下，不能在该元素的开始标签和结束标签之间嵌入脚本（参见 19.1 节）。也就是说，<script src="your-functions.js"> 此处输入其他功能 </script> 是无效的。对于任何 script 元素，要么仅通过 src 加载外部脚本，要么嵌入脚本并不含 src。

```
<!DOCTYPE html>
<html lang="en">
<head>
     <meta charset="utf-8" />
     <title>Loading an External Script</title>
     <!-- 在加载任何JS文件之前加载样式表 -->
     <link rel="stylesheet"
     → href="global.css" />

     <script>
     /*
     JavaScript代码写在这里
     */
     </script>
</head>
<body>
... 所有的HTML内容写在这里 ...
</body>
</html>
```

图 19.2.2　在这个例子中，嵌入脚本位于 head 中。它位于 link 元素的后面，从而让 CSS 文件更快地加载。关于为什么要尽可能地避免在 head 中嵌入脚本，参见 19.1 节的补充材料“脚本和性能的最佳实践”

19.3　JavaScript 事件

在本章的引言中，我提到对 JavaScript 的深入讨论超出了本书的范围。不过，我仍然想让你大体了解一下 JavaScript 事件，从而对 JavaScript 能做什么有一些基本的认识。

可以编写 JavaScript 对特定的、预定义的事件（由访问者或浏览器触发）进行响应。下面的列表只是编写脚本时可用的事件处理程序的一些简单示例。HTML5 引入了大量其他的事件处理程序，其中很多都是与 audio 和 video 元素相关的。有的触屏设备也开始支持特殊的基于触摸操作的事件处理程序。

注意下面的列表中的“mouse”（鼠标）代表所有“指针设备”。例如，onmousedown 会在访问者使用电子笔、真正的鼠标或其他类似的设备时被触发。

- onblur：访问者离开先前获得焦点的元素（参见 onfocus）。
- onchange：访问者改变元素的值或内容。通常用于表单字段（关于表单，参见第 16 章）。
- onclick：访问者点击特定的区域或在元素(如链接)获得焦点时按下回车键。
- ondblclick：访问者双击特定的区域。
- onfocus：访问者选择、点击或用制表键将焦点移至特定的元素。
- onkeydown：在指定的元素上，访问者按下一个键。
- onkeypress：在指定的元素上，访问者按下并松开一个键。
- onkeyup：在指定的元素上，访问者在输入后松开一个键。
- onload：浏览器完成页面的加载，包括所有的外部文件（如图像、样式表、JavaScript 等）。
- onmousedown：在指定的元素上，访问者按下鼠标键。
- onmousemove：访问者移动鼠标指针。
- onmouseout：访问者在鼠标指针停留的特定元素上移开鼠标。
- onmouseover：访问者将鼠标指向元素。
- onmouseup：访问者在点击元素后松开鼠标键（与 onmousedown 相反）。
- onreset：访问者点击表单的重置按钮或在该按钮获得焦点时按下回车键。
- onselect：访问者选择元素中的一个或多个字符。
- onsubmit：访问者点击表单的提交按钮或在该按钮获得焦点时按下回车键。

HTML5 事件处理程序的完整列表参见 http://dev.w3.org/html5/spec-author-view/global-attributes.html。一些触屏设备（如智能手机、平板电脑）包含的与触摸相关的事件处理程序包括 touchstart、touchend、touchmove 等（www.w3.org/TR/touch-events/）。

第 20 章

测试和调试网页

本章内容

- ❑ 验证代码
- ❑ 测试页面
- ❑ 尝试一些调试技巧
- ❑ 检查常见错误：一般问题
- ❑ 检查常见错误：HTML
- ❑ 检查常见错误：CSS
- ❑ 如果图像不显示

你编写好了全新的页面，却发现它们在浏览器中并没有像你预期的那样显示，或者完全不显示，或者在你的默认浏览器中显示得很好，但客户使用其他的浏览器打开时却发现页面显得有些奇怪。

在 HTML、CSS 和众多的浏览器（尤其是旧浏览器）和平台之间，很容易产生各种各样的问题。本章会提示一些常见的错误，并帮助你清除自己造成的错误。

这些调试技术中的一部分看起来相当基础，但网页的问题往往也非常基础。在查找大的问题之前，要确保没有任何小的问题。

无论如何，应该在一个或多个平台的几种浏览器中对网站进行充分的测试，查看各个页面是不是按照预期的方式工作（参见 20.2 节）。

20.1 验证代码

发现 HTML（如图 20.1.1 所示）和 CSS 中错误（如图 20.1.2 所示）的一种重要方法就是使用验证器，这样就不用你自己追踪了。HTML 验证器可以对代码和语言规则进行比较，并将其发现的不一致的情况显示为错误或警告。它还可以提示语法错误，无效的元素、属性和值，以及错误的元素嵌套（如图 20.1.3 所示）。它无法判断内容是不是由最能描述它的元素进行标记的，因此编写语义化的 HTML 还得靠你自己（参见 1.3 节）。

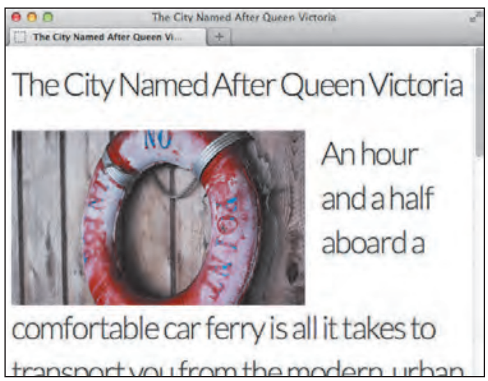


图 20.1.1　将需要检查的 URL 粘贴到 Address（地址）字段。选择 Show Source（显示源代码）选项，这样 HTML 源代码就会出现在验证器找到的错误的下方，有错误的 HTML 片段会突出显示

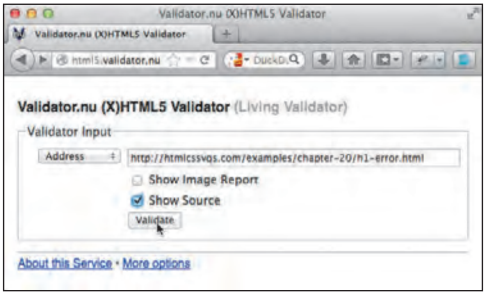


图 20.1.2　肯定哪个地方出了问题，标题下面的文本本不该这么大。我已经检查了 CSS，问题不是出于设置了很大的 font-size 值。问题出在哪儿呢



图 20.1.3　在第 10 行发现错误，原因是没有使用 </h1> 结束标签，错误地加上了另一个 <h1> 开始标签。其他的错误都是由第一个错误引起的，因此只要修复了这个错误，页面就没有任何错误了

将页面放到万维网上之前，不需要确保它们通过验证器的检查，完全没有错误。实际上，大多数网站都有一些错误。而且，W3C 的 CSS 验证器会将用于属性名称的厂商前缀标记为错误，但这并不意味着你应该将这些厂商前缀从样式表中移除（关于厂商前缀，参见第 14 章）。

浏览器可以处理很多类型的错误（同时忽略一些其他的错误），从而以它们能实现的最佳方式将页面呈现出来。因此，即便页面在验证时有错误，也可能看不出来。不过，有时错误会直接影响页面的显示（如图 20.1.1 所示）或行为。因此，应该使用验证器尽可能地排除代码中的错误。

验证器能找到的错误的示例参见 20.5 节和 20.6 节。

验证代码的步骤

(1) 首先使用 http://html5.validator.nu（如图 20.1.2 和图 20.1.3 所示）或 W3C 的 http://validator.w3.org 对 HTML 进行检查。更多信息参见头两条提示。

(2) 修复标出来的 HTML 错误，保存修改。需要的话，将文件再次上传到服务器，再重复第 (1) 步。

(3) 可以使用 http://jigsaw.w3.org/css-validator/ 检查 CSS 错误。类似地，修复你看到的错误，再重新检查页面。

检查 HTML 的一致性

对于 HTML 代码中某些部分的格式，HTML 是相当宽松的。例如，对于 img 这样的空元素，使用 /> 和 > 结尾都是合法的。HTML 验证器不会检查此类一致性问题。如果你希望代码一致，可以使用 HTML Lint（http://lint.brihten.com/html/）。通过它可以检查空元素是否闭合，开始和结束标签是否为小写字母，属性是否为小写字母等。

提 示　W3C 的验证器（http://validator.w3.org/）使用的是 http://html5.validator.nu/ 提供的验证引擎，因此使用这两种都可以。W3C 的错误消息更易读，但不会突出显示 HTML 源代码的错误部分。

提示　可以通过输入 URL（参见图 20.1.1）、上传 HTML 文件以及将 HTML 粘贴到验证器三种方式对 HTML 进行验证。如果使用上传或复制粘贴的方式，不必将文件上传到服务器就可以进行检查了。

提示　一个 HTML 错误可能导致多个验证器报错结果。例如，缺少一个结束标签会导致多条错误消息（参见图 20.1.3）。如果修复了这个结束标签的问题，所有这些后续错误就都不存在了（如图 20.1.4 所示）。因此，应该按照从上往下的顺序，一次修复少量错误后就立刻再次验证，看看其他的问题是否也已解决。

提示　如果样式表中包含第 11 章介绍的 `.clearfix` 样式规则，CSS 验证器会显示几个错误。这是由于使用了为旧版本的 Internet Explorer 准备的非标准的 CSS。你可以忽略这些错误，它们不会影响你的页面。

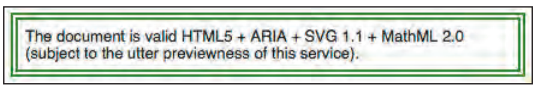


图 20.1.4　通过验证器的检查，HTML 是合法的



图 20.1.5　刷新页面后可以看到页面好看多了

20.2　测试页面

即便代码通过了验证，页面可能仍然不像预期的那样工作，如图 20.2.1 ～图 20.2.3 所示，或者可能在一个浏览器中是正常工作的，在另一个浏览器中却有问题。你并不知道用户使用哪种浏览器，因此在不同的浏览器上对页面进行测试是很重要的（参见本节末尾的补充材料“关于浏览器测试的更多信息”）。

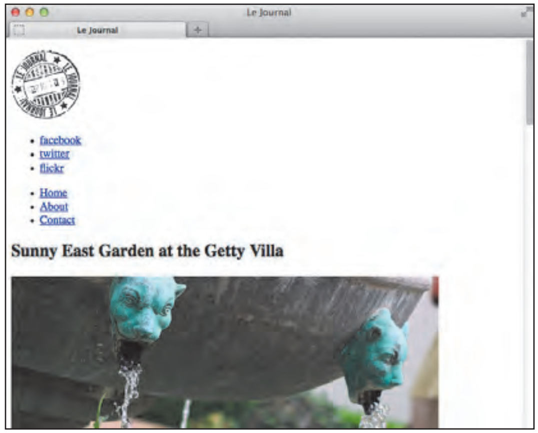


图 20.2.1　这个页面通过了验证，但它看上去并不像预期的那样。问题出在哪儿（参见图 20.6.2）

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Le Journal</title>
    <link rel="stylesheet"
    → href="css/style.css" />
    ...
</head>
<body>
...
</body>
</html>
```

图 20.2.2　问题出在指向 CSS 文件的链接上——CSS 文件的名称为 styles.css，而这里指向的却是 style.css。浏览器无法找到 CSS，因此对页面的显示是有问题的

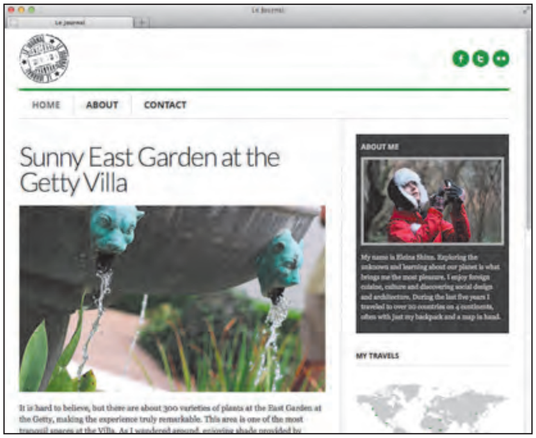


图 20.2.3　改正了代码中的文件名后，样式表就被加载进来了。现在页面可以正确显示了

测试 HTML 页面的步骤

(1) 对 HTML 和 CSS 进行验证（参见 20.1 节），作出必要的修改。

(2) 按照 2.7 节的解释打开页面。

(3) 检查整个页面，确保与你希望看到的完全一样。例如：

- ❑ 格式与期望的是否一致
- ❑ 链接的 URL 是否指向了正确的页面或资源（可以通过激活链接并查看结果对 URL 进行测试）
- ❑ 所有的图像都出现了吗？它们的位置和对齐方式是对的吗？
- ❑ 如果你检查的是响应式网页，网页布局是否能适应不同的屏幕尺寸

(4) 在不关闭浏览器中页面的情况下，打开有关的 HTML 或 CSS 文档，作出必要的改动。

(5) 保存修改。

(6) 切换到浏览器，刷新页面并查看所作的改动。

(7) 重复第 (3) 步至第 (6) 步，直到你对网页满意为止。有时候可能需要测试很多次才能解决问题。如果尝试很多次后仍然失败了，请再次验证代码，确保没有引入新的错误。

(8) 从第 (2) 步开始，在其他的浏览器中执行同样的测试流程，直到满意并认为页面做好了发布准备为止。

(9) 将文件上传至服务器。

(10) 回到浏览器，在地址栏中输入页面的 URL，按下回车键。页面将出现在浏览器中。

(11) 再次对页面进行检查，确保没有任何问题（很容易忘记上传图像或者页面需要的其他文件）。同时，如果访问者可能通过移动设备访问网站的话，别忘了在移动设备上对其进行检查。

测试工作流

一种常见的测试工作流就是在构建页面的过程中定期在几个浏览器中对页面进行检查。当页面完成以后，再在全套浏览器中检查，对代码作必要的修改。完成整个网站之后，可能还需要在浏览器中再次进行测试，确保整个网站没有问题。

我建议首先对网站的本地版本进行测试和改进（参见“测试 HTML 页面的步骤”中的第 (1) ~ (8) 步），这是在文件上传到服务器上（参见第 (9) 步）之前进行的。将其上传到服务器上之后，再对它们做完整的检查，不过针对的是服务器上的文件。无论你对本地文件做了多少次检查，都不要跳过这一步，因为服务器上的网站才是访问者使用的网站（参见第 (10) ~ (11) 步）。

提 示　如果浏览器中显示的是 HTML 代码而不是页面，要确保文件使用的是 .html 或 .htm 扩展名（而不是像 .txt 这样的扩展名）。

提示　有时候，页面上的问题并不是你的错，尤其是样式上的问题。在认为问题出在代码上之前，一定要确保浏览器对遇到问题的特性是支持的。附录 A 和附录 B 中有关于 HTML 和 CSS 特性的浏览器支持情况的资源链接。关于浏览器支持信息，Can I Use（http://caniuse.com）和 Quirksmode（www.quirksmode.org/css/）是两处非常宝贵的资源。

应该在哪些浏览器中进行测试

通常，大多数网站开发人士会在以下浏览器中对网站进行验证。

- Chrome（www.google.com/chrome）的最新版本。Chrome 会在你的计算机上自动进行更新，大约每隔六周就会发布一个新的版本。
- Firefox(www.firefox.com)的最新版。Firefox 同 Chrome 一样，更新和发布频率也非常快。
- Internet Explorer 8+，只针对 Windows。（IE8 的市场份额会越来越少，最终可能会从你的验证清单中消失。）可从 www.microsoft.com 下载 IE 的各个版本。
- Safari（www.apple.com/safari/）的最新版，有时候是 Safari 5+。Safari 预装在 OS X 中。

至于 Opera，开发人员通常会选择测试它，Opera 在世界上很多地区都有少量的市场份额。Opera 的下载地址为 www.opera.com/。

好消息是，在不考虑 IE8 甚至 IE9 的情况下，这些浏览器对 HTML 和 CSS 特性的支持能力几乎在同一个水平。这意味着你通常很难看到不同浏览器之间的差异，除非你的页面用到了特别新的 HTML5 或 CSS3 特性。因为 IE8 已经很旧了，因此如果你的网站在 IE8 中和在现代浏览器中的样子稍有出入，也是可以接受的。

获取测试用的浏览器

可以使用上面提供的链接下载并安装这些浏览器，但如果你使用的是一台 Mac，该如何测试 IE 呢？如果你使用的是 Windows，如何测试 Safari 以及不同版本的 IE 呢？这里给出了一些方案。

- 虚拟机（VM）是运行在你的计算机上的一个独立的操作系统。微软为不同版本的 Windows 和 IE 提供了虚拟机，因此你可以在 Mac、Windows 或 Linux 机器上对 IE 进行测试。这些虚拟机位于 www.modern.ie/en-US/virtualizationtools#downloads。不能在一台 Windows 机器上运行 Mac 虚拟机以测试 Mac 浏览器。
- 使用 BrowserStack（www.browserstack.com）和 Sauce Labs（http://saucelabs.com）可以在很多浏览器和移动设备上免费测试你的页面。

在手机和平板电脑上测试

你可能至少希望在 iOS 和 Android 设备上进行测试。测试页面的移动兼容性是很有挑战的，因为通常很难在移动设备上对网站进行修改。解决的办法包括使用 BrowserStack 和 Sauce Labs（注意，使用模拟器并不一定与实际设备上的效果完全一致）。

- 使用苹果的 iOS Simulator 测试页面在 iPhone 和 iPad 上的情况。这种方法最大的局限性在于该模拟器只能在 OS X 上运行，在 Windows 上没有可替代的软件。iOS Simulator 是免费的 Xcode 的一部分，下载地址为 http://developer.apple.com/xcode/。
- DeviceAnywhere（www.deviceanywhere.com）提供了一种在线访问各种不同移动设备进行测试的收费功能。DeviceAnywhere 也有免费版本，不过一次只能用 10 分钟。
- 使用 Electric Plum 针对 Windows 的 iPhone 和 iPad 模拟器（www.electricplum.com）。这不是由苹果提供的，因此与苹果的 iOS Simulator 不一样。
- 使用为其他设备和移动浏览器准备的模拟器。Mobile Boilerplate 维护了一个列表，见 https://github.com/h5bp/mobile-boilerplate/wiki/Mobile-Emulators-&-Simulators。
- 查看 Open Device Lab（http://opendevicelab.com），看看你身边是否有可供免费测试的设备。
- 如果你有设备，结合使用 Adobe Edge Inspect（http://html.adobe.com/edge/inspect/）可以简化测试和修正问题的过程。

浏览器市场变化很快：当你阅读本书的时候，这些浏览器又会有一些新的版本发布，还会冒出一些新的设备。不过，只要遵循渐进增强的原则，你的网站就可以为旧的浏览器提供简单的体验，为现代浏览器提供增强的体验。

20.3 尝试一些调试技巧

你现在应该已经做了一些测试，也发现了一些 bug。下面是用于排查网页错误的一些真实技巧。

- 首先检查常见的错误。
- 逐步开展工作。做出小的改动，并在每次改动后进行测试。这样才能在问题出现之后定位到问题的来源。
- 调试时，从你能确保正确的地方开始。之后再逐一添加可能出问题的部分，每次添加完以后都在浏览器中进行测试，直到找到问题的来源为止。
- 根据前面的要点，用排除法寻找产生问题的代码片段。例如，可以注释掉一半代码，检查问题是否出于另一半代码，如图 20.3.1 所示。再在有问题的代码中注释掉更小的部分，直到找出问题。（参见 3.16 节和 7.2 节。）
- 留意输入错误。很多令人费解的错误是由简单的输入错误造成的。例如，在 HTML 中以某种方式拼写类名，却在 CSS 中使用了另一个名称。
- 在 CSS 中，如果不确定问题是出在属性上还是出在选择器上，可以试着在选择器上添加极其简单的声明，如 `color: red;`、`border: 1px solid red;` 等（如果 `red` 已经用于网页的设计，也可以选择一种不常见的颜色，如 `pink`）。如果元素变成红色，那么问题就出在属性上，否则问题就出在选择器上（假定不存在另一个特殊性更强的选择器，也不存在比当前选择器更靠后的选择器）。
- 使用一个或多个开发工具，直接在浏览器中对 HTML 或 CSS 进行修改测试。或者使用这些工具审查代码，试着对问题进行定位。（参见补充材料“浏览器开发工具”。）

20

```
...

.excerpt {
    border-top: 1px dotted #ccc;
    margin: 0 .5em 2em 0;
}

.excerpt .title {
    font-size: 1.25em;
    line-height: 1.2;
}

/*
.more,
.excerpt .date {
    text-align: right;
}

.excerpt .date {
    line-height: 1;
    margin: 0 1em 0 0;
    padding: 0;
    position: relative;
    top: -1em;
}
*/

.photo {
    float: left;
    height: 300px;
    width: 400px;
}

...
```

图 20.3.1　我注释掉了这段代码中间的一部分（位于 /* 和 */ 之间），以查看它是不是出错的原因。注意，很多 HTML 和 CSS 编辑器都包含代码高亮，即自动为元素、选择器等设置不同的颜色。这对调试是有帮助的。例如，如果将 CSS 属性名称打错了，编辑器就不会以预期的颜色显示它，表明它是无效的

浏览器开发工具

浏览器都包含帮助你调试页面或者具有其他功能的工具。你会发现，一种反复用到的特性就是直接修改 CSS 或 HTML 并立即查看它们对页面的影响。这可以让你快速地对修改进行测试，满意之后再将它们包含到代码里去。

下面是各个浏览器中最常用的工具。

- Chrome：DevTools（http://developers.google.com/chrome-developer-tools/）。
- Firefox：Firefox 有内置工具（https://developer.mozilla.org/en-US/docs/Tools），但是 Firebug 插件特别流行（http://getfirebug.com）。此外，Web Developer（http://chrispederick.com/work/web-developer/）是稍有区别的一种工具，但也非常好用。在同一个链接的页面上，也可以下载用于 Chrome 的版本。
- Internet Explorer：F12 Developer Tools（http://msdn.microsoft.com/en-us/library/hh772704(v=vs.85).aspx）。
- Opera：Dragonfly（www.opera.com/dragonfly/）。本书写作之时，这个特性还没有成熟。
- Safari：Web Inspector（http://developer.apple.com/technologies/safari/developer-tools.html）。

网上还有演示如何使用这些工具的文档和视频。在 20.6 节有使用 Chrome DevTools 的示例。

20.4　检查常见错误：一般问题

浏览器之间的差异**可能是**由一些不明显的浏览器漏洞造成的，也可能是由于使用新技术造成的，但更常见的则是出于一些简单的问题。

每个人在从新手到专家的道路上难免会犯一些低级错误。例如，你有可能认为问题出在代码上，从而花了大量时间进行调试，最后才发现你修改的是一个文件，上传和查看的却是服务器上的另一个文件！

下列建议中的大多数适用于使用网站在服务器上的 URL 对网站进行测试的情形。

查检笼统的常见错误

- 根据 20.1 节的描述，对代码进行验证。这是一个很好的起点，因为这样就能排除代码语法相关的错误了。
- 确保已上传要测试的文件。
- 确保上传的文件的位置是正确的。
- 确保输入的 URL 与要测试的文件是对应的。如果要查看的页面是从另一个页面跳转过来的，确保链接中的 URL 与页面的文件名和位置是完全匹配的。
- 确保上传文件之前已经进行了保存（包括最新修改）。
- 确保上传了所有辅助文件（包括 CSS、图像、音乐、视频等）。
- 确保 URL 的大小写与文件名的大小写是完全匹配的。（这也是推荐全部使用小写字母的原因，这样做可以降低输入 URL 时产生错误的可能性——既针对网站开发人员，也针对访问者。）确保文件名中没有使用空格（应使用短横线代替空格分隔单词）。
- 如果在以前的测试中曾禁用某项浏览器功能（如 JavaScript 支持），确保重新启用这些功能。
- 确保问题不是出自浏览器本身。对于这一点，最简单的方法就是换个浏览器对页面再测试一遍。

在接下来的两节里，我会介绍如何检查 HTML 和 CSS 中的常见错误。

20.5　检查常见错误：HTML

有时，问题出现在 HTML 中。

检查 HTML 常见错误的方法

- 很容易出现一两个输入错误，如图 20.5.1 所示。确保所有的拼写都是正确的，属性的值都是有效的，如图 20.5.2 所示。使用 HTML 验证器可以查出这类错误，从而能迅速地改正（参见 20.1 节）。
- 留心元素的嵌套。例如，如果先开始 `<p>`，再使用 `<em>`，就要确保 `</em>` 结束标签位于最后的 `</p>` 之前。

还是有问题

如果你阅读完本章，发现还是有一些问题无法解决，这里还有一些其他的建议。

- 当我建议你休息一下的时候，不要认为我的建议是出于傲慢。有时候，你能做出的最佳选择就是暂时将问题搁到一边。当你回头再处理时，答案可能就在眼前。相信我，我有过这样的经历！
- 回到页面能正确工作的最近的版本。（为此，要在建立页面的过程中保存页面的副本，从而在必要的时候能回到历史版本。）然后再一点一点地增添新的元素，并随时进行测试。
- 对于页面链接到的资源（如 CSS、图像、JavaScript 或媒体文件），直接在浏览器的地址栏中输入资源的 URL，确保该资源确实是存在的。
- 还有很多网站用于搜索问题的答案。Stack Overflow（www.stackoverflow.com）和 SitePoint（www.sitepoint.com/forums/）便是两个例子。你还可以搜索到一些其他类似的网站。

20

- 如果重音字符或特殊符号没有正常显示，要确保文档 head 元素开始后有 <meta charset="utf-8" /> 语句（如果不使用 UTF-8，也可以使用其他恰当的字符编码），同时确保文本编辑器是使用与之相同的编码保存 HTML 文件的。如果还有问题，试着使用恰当的字符引用。
- 确保属性值是用直引号而不是曲引号包围的。如果属性值是用双引号包围的（这也是惯例），则属性值中可以有单引号，如图 20.5.3 所示。如果属性值本身包含双引号，则应使用字符引用，如图 20.5.4 所示。
- 不要对空元素使用分开的开始标签和结束标签，如图 20.5.5 所示。（从技术上说，即便省略结束标签，或者对空元素添加结束标签，浏览器也能正确地显示，但最好还是严谨一些。）

```
<img scr="woody.jpg" width="200px"
→ height="150px" alt="Woody the cat" />
```

图 20.5.1　你能找出问题吗？我将 src 拼错了，还在 width 和 height 的值中使用了单位。如果你没有注意到这些错误，可以使用 HTML 验证器。它们可以标出这类输入错误，从而节省发现这些错误的时间

```
<img src="woody.jpg" width="200" height="150"
→ alt="Woody the cat" />
```

图 20.5.2　在正确的版本中，src 的拼写是正确的，width 和 height 的值中也去掉了 px

```
<img src="jungle.jpg" width="325" height="275"
→ alt="Llumi's jungle" />
```

图 20.5.3　如果属性值包含单引号，可以照常用双引号包围属性值

```
<img src="cookie-the-cat.jpg" width="250"
→ height="200" alt="Cookie's saying,
→ &quot;Enough!&quot;" />
```

图 20.5.4　如果属性值包含双引号，属性值里的双引用号就应使用字符引用

```
<img src="jungle.jpg" width="325" height="275"
→ alt="Llumi's jungle"></img>
```

图 20.5.5　不要对空元素（如 img）包含结束标签，HTML 验证器会将这个例子标记为错误

20.6　检查常见错误：CSS

尽管 CSS 的语法相当简单明了，但它也有一些常见的陷阱，特别是如果你习惯了编写 HTML 的话。CSS 验证器可以将如本节讨论的这类错误标示出来，因此，在查看 CSS 以寻找错误之前，要先对样式表进行验证（参见 20.1 节）。

检查 CSS 常见错误的方法

- 确保使用冒号（:）分隔属性和值，而不是像在 HTML 中那样使用等号，如图 20.6.1 和图 20.6.2 所示。

```
p {
    font-size=1.3em;
}
```

图 20.6.1　哎呀，改变用等号分隔属性和值的习惯很难

```
p {
    font-size: 1.3em;
}
```

图 20.6.2　好多了。属性和值之间应该使用冒号。在冒号之前或之后添加额外的空格不会产生影响，但通常在冒号之后加上一个空格

- ❑ 确保使用分号（;）结束每个属性 – 值对（即**声明**）。确保没有多余的分号，如图 20.6.3 和图 20.6.4 所示。

```
p {
    font-size: 1.3em font-style: italic;;
    → font-weight: bold;
}
```

图 20.6.3　又错了。每个属性 – 值对之间必须有且只能有一个分号。这里，有一处少写了一个分号，另一处多写了一个分号

```
/* 还是有问题，但更容易发现问题了 */
p {
    font-size: 1.3em
    font-style: italic;;
    font-weight: bold;
}

/* 这是正确的版本 */
p {
    font-size: 1.3em;
    font-style: italic;
    font-weight: bold;
}
```

图 20.6.4　如果每个属性 – 值对都独占一行，错误就更容易发现了，因为这样分号就不会混在一大片属性、值和冒号之中了

- ❑ 不要在数字和单位之间添加空格，如图 20.6.5 和图 20.6.6 所示。

```
p {
    font-size: .8275 em;
}
```

图 20.6.5　又错了。不要在数字和单位之间添加空格

```
p {
    font-size: .8275em;
}
```

图 20.6.6　这次对了。注意冒号和值之间的空格是可选的（但通常包含这个空格）

- ❑ 不要忘了后括号。
- ❑ 确保使用可接受的值。像 `font-style: none;` 这样的声明是无效的，因为该属性的空值为 `normal`。
- ❑ 使用嵌入样式表时，不要忘了 `</style>` 结束标签（无论如何，在多数情况下你都应该避免）。
- ❑ 确保 HTML 文档正确地指向 CSS 文件。
- ❑ 留意 CSS 选择器之间的空格和标点符号。
- ❑ 确保浏览器支持你编写的代码，尤其是最新特性，因为浏览器在 CSS3 成长的过程中也在不断演变。关于浏览器对新特性的支持情况参见 Can I Use（http://caniuse.com）或者 Quirksmode（http://www.quirksmode.org/css/）。CSS 验证器无法判断浏览器是否支持某个 CSS 特性，但如果输入的选择器、属性或值是 CSS 中不存在的，验证器仍会给出提示。
- ❑ 使用浏览器开发工具审查浏览器所解析的样式规则，从而快速地查出哪些代码没有按照预期进行解析，或者查看所应用的特殊性规则，如图 20.6.7 所示（参见 20.3 节补充材料“浏览器开发工具”）。

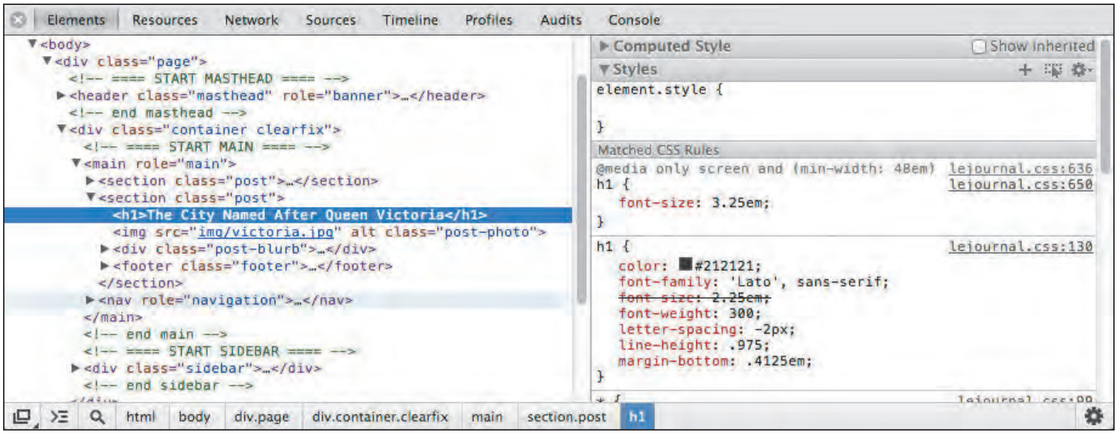


图 20.6.7　这里使用 Chrome 的 DevTools 对代码片段 `<h1>The City Named After Queen Victoria</h1>` 进行了审查。HTML 显示在左侧面板，应用于该元素的 CSS 显示在右侧面板。其他浏览器的工具跟 DevTools 配置相似。`font-size` 设置上有一条删除线，表示这条声明被另一条规则（显示在媒体查询上面的那条）覆盖了。这样的结果是我在这个例子中想要的效果，但你可以使用这种技术跟踪样式没有按照预期进行应用的原因。还可以编辑 HTML 和 CSS 规则，并直接在浏览器中测试修改。所有的浏览器开发工具都允许这种行为。如果你对结果满意，就可以对 HTML 和 CSS 文件作出真实修改

20.7　如果图像不显示

小红叉、碎片图标、替代文本，或者什么都不显示——这都是图像未能正确加载的标志，如图 20.7.1 和图 20.7.2 所示。

```
...
<body>
<h1>The City Named After Queen Victoria</h1>

<img src="img/Victoria.jpg" width="370"
→ height="220" alt="A weathered life ring
→ indicates you're headed toward another time
→ and place." />

<p>An hour and a half aboard ...</p>
</body>
</html>
```

图 20.7.1　图像的文件名为 victoria.jpg，但在 HTML 中，引用的却是 Victoria.jpg（以大写字母 V 开头）。因此，在服务器上检查页面时，图像不会显示



图 20.7.2　在你的计算机上，如果系统不区分文件名的大小写，页面看起来就没有问题。但将页面发布到区分大小写的服务器上以后，便无法找到图像。像 Chrome 这种浏览器就仅会显示一个破碎图标

修复缺失的图像

- ❑ 首先检查图像在服务器上的文件名是否与 `img` 元素中引用的名称严格匹配，

包括大小写、扩展名，如图 20.7.1 所示。不要在文件名中包含空格，参见 1.6 节。

- 确保 `img` 元素的 `src` 属性中的图像 URL 是正确的。一种简单的测试方法就是将图像放到 HTML 页面所在的目录。这样就只需要 `img` 元素中的文件名和扩展名正确就可以了，无需引入路径信息。如果图像显示出来了，问题很可能就出在 `src` 上。不过，将图像放在 HTML 文件所在的目录并不好，因为这样的话，网站很快就会变得无序。因此，在测试之后，要将图像从 HTML 页面目录移出，并修改指向图像的 `src` 路径。参见 1.7 节。
- 如果在你的计算机上查看页面时图像是显示的，而将页面上传至服务器之后，图像却没有显示出来，要确保已经将图像上传到了服务器。
- 把图像保存为 PNG、JPEG 或 GIF 格式了吗？如果是的话，所有的浏览器都能显示该图像；如果是其他格式，那就很难说了。更多信息参见第 5 章。

20

第 21 章

发布网站

21

本章内容

- ❑ 获得域名
- ❑ 为网站寻找主机
- ❑ 将文件传送至服务器

当你完成自己的杰作并准备好将它展现给公众时，必须将页面传送到万维网上，人们才能看到这些页面。本章介绍了所需的步骤：获取域名，寻找 Web 主机，将域名与主机连接起来，以及将文件上传到主机的服务器上。

如果你希望更改 Web 主机，可以将网站从一个服务器转移到另一个上面。你的域名及整个网站的 URL 可以保持不变。

在发布页面之前和之后，要确保对页面进行彻底的测试。更多细节信息参见第 20 章。

21.1　获得域名

在访问者能够看到你的网站之前，需要为网站关联一个域名，如图 21.1.1 所示。只要你按照本章所示的步骤操作，任何人都可以在浏览器中访问这个域名。



图 21.1.1　只有特定的公司才是可信的域名注册商（这个界面和下面的界面均来自 Namecheap）。你可以使用其中的一个查询你想使用的域名是否可用，也可以通过 Web 主机提供商的网站进行检查

获得域名的步骤

(1) 在浏览器中访问某个域名注册商，查看你想用的域名是否可用，如图 21.1.2 所示。（并参见第一条提示。）如今，要想找一个还未被使用的名字非常不容易，因此你有必要搜索一下你喜欢的名字的变体。

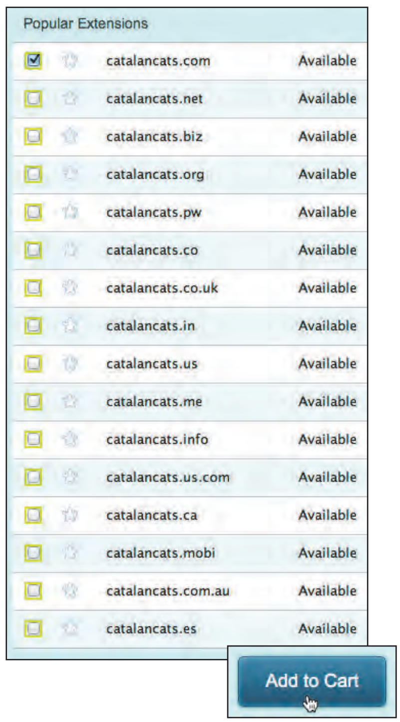


图 21.1.2　如果域名是可用的，就可以通过第三方注册商网站或者 Web 主机提供商进行注册（从图中可以看到，catalancats.com 这个域名是可用的）

(2) 一旦找到一个域名，便可以注册了。不同域名注册商的价格不一样，不过 .com 域名的价格通常大约为每年 10 美元（其他类型的域名的价格可能不一样）。

提示 Namecheap（www.namecheap.com）和 Hover（www.hover.com）只是可以注册域名的网站中的两个(没有要求必须使用它们)。通过搜索“domain registrars”（域名注册商）可以找到其他一些类似的网站。此外，很多 Web 主机提供商也提供域名注册服务；反过来，很多域名注册商也提供主机服务。域名注册和主机托管可以使用不同的提供商，很多人都这样做。

提示 要让网站在别人访问 URL 时显示出来，需要先进行一些重要的配置，这些内容参见下一节的补充材料“连接域名和 Web 主机”。

21.2 为网站寻找主机

Web 主机提供商可以提供其服务器上的一部分空间，存放你的网站文件，同时提供其他相关的服务，如创建同域名关联的电子邮件地址（如 *yourname@yourdomain.com*）。

有很多公司都提供网站托管服务。其中，大部分公司根据其服务的内容按月收取费用。有的公司提供免费的网站托管服务，但要求在你的网站上放置他们的广告（一般不推荐这种方法）。

尽管你可以在互联网上搜索 Web 主机提供商，但还是推荐你向使用某种主机的朋友进行咨询——抑或某位值得信赖的博主提到的他所用的主机的提供商。另外，登记一下网站的页脚和附注栏。

考虑主机提供商时，除了价格之外，还要考虑以下几件事件。

- 他们允许一个主机使用多个域名吗？是否需要为每个网站支付额外的费用？如果你希望自己的网站未来发展得很好，就应该寻找可以使用多个域名的主机。
- 他们允许网站使用多大的磁盘空间？不要为超出需要的部分付费。话虽如此，通常连最基本的账户也拥有足够的空间。记住，HTML 文件占用的空间非常小，而图像、音频和视频文件则依次占用更大的空间。如果有必要，使用一段时间后，你总是能够升级到拥有更多空间的账户。
- 他们允许每个月使用多大的数据传输量（带宽）。这个值代表的是发送给访问者的数据（包括 HTML、CSS、图像、媒体文件等）的总大小，而不是允许在服务器存储的空间的大小。因此，如果你预计访问者会从你的网站访问大量的大文件，就需要更大的月传输配额。关于这一点，通常来说，网站刚起步时，最基本的账户就能满足需求，将来可以视情况再升级。
- 可以使用域名创建多少邮箱（主机提供商通常提供很多个。）
- 他们有应对大访问量、防止网站崩溃的措施吗？
- 他们提供哪种技术支持？是通过电话、邮件、还是在线交谈，他们响应请求需要多长时间，他们的网站上有大量的支持信息吗？（在成为他们的客户之前，可以检查这些内容的数量。）
- 他们多久备份一次服务器上的数据(以防出现问题)？
- 可以使用哪些服务器端语言和软件包？多数至少会提供 PHP、MySQL，

还有一些会提供 WordPress 或其他高级特性。有些特性需要更昂贵的付费方案。

将 ISP 作为 Web 主机提供商

如果你可以访问互联网，就可能已经通过你的 ISP（Internet Service Provider）获得了少量的 Web 空间。它可能难以存放整个网站，但放几个页面倒是足够。可以向你的 ISP 询问详细信息。

不过，要记住的是，这类托管空间通常不允许使用单独的域名访问网站，例如，可以使用 www.someisp.com/your-site/，但不能使用 www.yourdomain.com。因此，如果你希望建设的是专业的网站，就不要使用 ISP 提供的免费空间存放网站。

Web 分析

Web 分析是一份报告，告诉你有多少人访问过你的网站，他们使用了哪些浏览器，哪些页面是最受欢迎的，以及很多其他有用的数据。

你所用的 Web 主机有可能提供这些信息，但如果你希望获得更为丰富的信息，可以使用 Google Analytics 或类似的服务。该工具很容易添加到网站，只要在每个页面中添加一小段代码就可以了。更多信息参见 www.google.com/analytics/。

连接域名和 Web 主机

注册了域名，找到了 Web 主机之后，下面很重要的一步就是将二者结合到一起：你必须将你的域名指向你的 Web 主机，这样，访问者输入网站的 URL 时网站才能加载。

要实现这一步，需要对你的域名配置域名服务器。Web 主机提供商会提供用于这项配置的域名服务器信息。

根据注册域名的方式（参见 21.1 节）的不同，实际的配置是在以下两个地方中的一个进行的。如果域名是通过域名注册商注册的，可以登录到对应的账户，为域名设置域名服务器信息（域名注册商会提供操作说明）。如果域名是通过 Web 主机提供商注册的，可以登录到对应的账户，更新设置。

如果这些内容让你感到有些疑惑，也不必担心。Web 主机提供商和域名注册商会提供操作上的说明，通常，如果需要，他们还能提供手把手的帮助。

需要记住的是，当你修改域名服务器设置时，通常需要 24 至 48 小时（最多 72 小时）的时间，相应的更新才能传播到整个万维网。不过，这种改变不会在所有的地方同时发生。因此，如果你更新了域名服务器信息（并按照 21.3 节的说明上传了网站的文件），你的朋友可能已经可以在他所在的地方正常访问网站了，而你却无法立即看到网站（也可能是相反的情况）。你的网站应该在不久之后就能让所有人都看到。

21.3　将文件传送至服务器

为了让互联网上的人们看到你的页面，你需要将页面上传到 Web 主机服务器。一种简单的方法就是使用 FileZilla（http://filezilla-project.org）这样的 FTP 客户端。FileZilla 是一款自由软件，可以运行在 Windows、Mac OS X 和 Linux 上（关于其他的 FTP 客户端，

参见提示）。很多网页编辑器也包含 FTP 功能，从而可以在编辑器中发布页面，而不必使用 FileZilla 这样的程序。

通常，你的 Web 主机提供商会在你注册托管账户之后通过电子邮件向你发送 FTP 连接信息。（如果你没有收到，可以联系他们。）一旦获得这些信息，就可以配置服务器连接，并将该配置保存在一个名称下面，如图 21.3.1、图 21.3.2 和图 21.3.3 所示，从而在以后想要发布文件（或从服务器上下载文件）的时候可以方便地访问服务器。

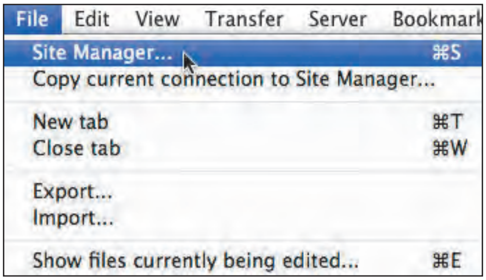


图 21.3.1　如果要在 FileZilla 中输入新的服务器信息，从主窗口中选择 File → Site Manager（文件→站点管理器）。Site Manager 是为每个站点配置 FTP 连接信息的地方

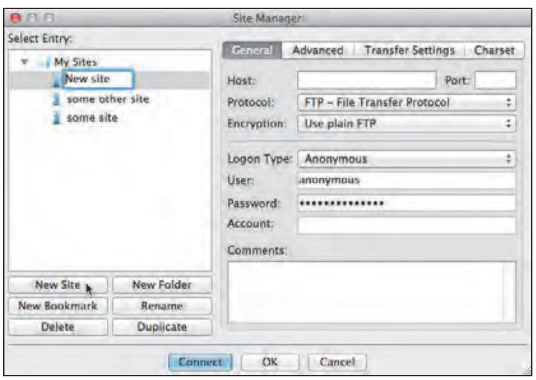


图 21.3.2　在 Site Manager 中点击 New Site（新站点）按钮，My Sites（我的站点）下面就会出现一个临时的名称

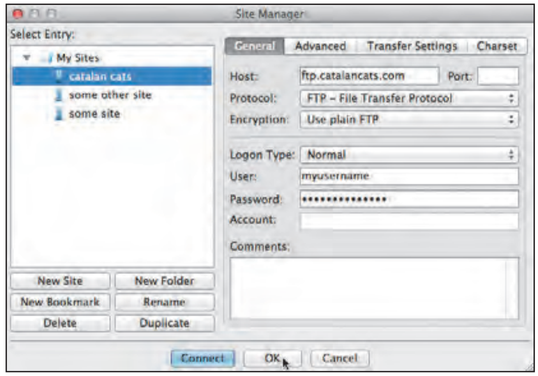


图 21.3.3　使用你选择的名称替换临时名称，再在 General（常规）标签页中配置连接信息。点击 Connect（连接）按钮，会保存这些信息并立即建立与服务器之间的连接。点击 OK（确定）按钮，则仅会保存信息

接下来，连接到服务器（如图 21.3.4 所示）和传输文件（如图 21.3.5 和图 21.3.6 所示）的操作就相当简单了。

注意，FileZilla 在 Mac OS 和 Windows 上的外观有些不同，但其界面是非常相似的（截图来自不同的操作系统）。除非特别指出，以下使用步骤对两个版本来说都是相同的。

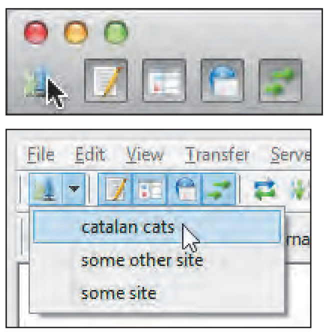


图 21.3.4　当站点的连接信息保存到 Site Manager 之后，就可以在不重复输入的情况下直接连接到 Web 主机的 FTP 服务器。在 Mac 或 Windows 上，通过这里显示的服务器图标回到 Site Manager（也可以通过图 21.3.1 中的菜单）。在 Windows 上，就像这里底部图像显示的，可以激活向下的箭头，再从弹出的下拉菜单中选择站点名称

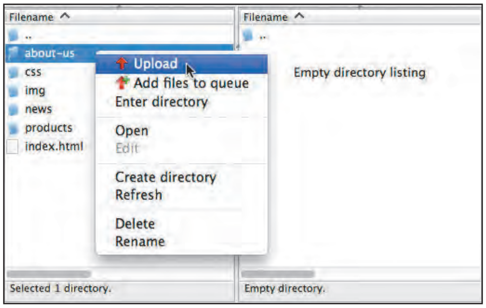


图 21.3.5　在窗口左侧，定位到你的计算机上包含要上传的文件的目录。在窗口右侧，选择服务器的目标目录。再在要上传至服务器的文件或文件夹上点击鼠标右键，选择 Upload（上传）

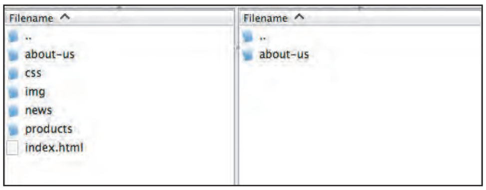


图 21.3.6　新上传的文件夹出现在窗口右侧的框内。对所有要传送至服务器的文件和文件夹执行相同的操作。如果要一次性上传多个文件和文件夹，可以先选中这些文件和文件夹，再点击鼠标右键并选择 Upload

1. 定义新的 FTP 站点的属性

(1) 在 FileZilla 的主菜单中选择 File → Site Manager（如图 21.3.1 所示），或者点击服务器图标（如图 21.3.4 所示），显示 Site Manager 窗口。

(2) 在 Site Manager 窗口中，点击 New Site（新站点）按钮（如图 21.3.2 所示）。

(3) 输入站点名称（替换临时名称）。这个名称不必与域名相同，它只是一个标签。根据你的 Web 主机提供商的说明，填写 General 标签页下各字段的信息。通常，至少需要输入主机 URL，在 Logon Type（登录类型）中选择 Normal（常规），输入用户名和密码（通常在建立账号时创建），如图 21.3.3 所示。

(4) 输完连接信息之后，点击 Connect 按钮保存信息并立即连接到服务器，或者点击 OK 按钮保存信息供以后连接用，如图 21.3.3 所示。

2. 使用 FileZilla 将文件传送至服务器

(1) 打开 FileZilla。

(2) 选择最左端的服务器图标（如图 21.3.4 所示），显示 Site Manager 窗口。再在弹出的下拉菜单中选择你的站点名称，点击 Connect 按钮（Windows 版有捷径，如图 21.3.4 所示）。FileZilla 就会建立与服务器的连接。

(3) 在窗口的右侧，定位到要上传文件的服务器目录。

(4) 在窗口的左侧，定位到你的计算机上存放要上传的文件的目录。

(5) 在左侧框右击目标文件或者文件夹，再在弹出的菜单中选择 Upload（如图 21.3.5 所示），文件就会进行传输，如图 21.3.6 所示（对于视频等大文件，这一过程所需要的时间会长一些）。还可以反过来传输文件（参见第一条提示）。也可以从一端将文件拖放到另一端，而不用右击的办法。

(6) 你对站点的更新现在已经能在线上看到了。通过 *www.yourdomain.tld*（这里的 *yourdomain.tld* 是你注册的域名；其中，*.tld* 是顶级域名，一般为 .com，但也可以注册使用其他扩展名的域名）访问你的站点，确保一切都是正常的。如果需要，编辑你的计算机上的文件，再按照第 (3) 步至第 (5) 步的说明将它们上传至服务器（如果过去了很长时间，则还需执行第 (2) 步）。重复这一步，直到网站完全符合你的预期。

(7) 完成文件传输之后，可以关闭 FileZilla 或在主菜单中选择 Server → Disconnect（服务器→断开连接），如图 21.3.7 所示。

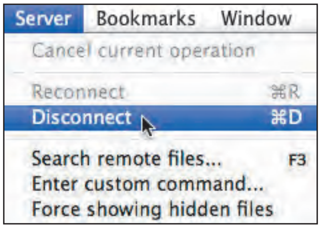


图 21.3.7　完成之后，选择 Server → Disconnect

提 示　还可以将文件从网站服务器传至你的计算机。要实现这一步，只需要右键单击右侧框中的文件或文件夹（如图 21.3.5 所示），再在弹出的菜单中选择 Download（下载）。

提 示　FileZilla 只是众多 FTP 客户端中的一个。CyberDuck（自由软件，http://cyberduck.ch）在 OS X 和 Windows 上均可用，OS X 上的流行软件还有 Transmit（www.panic.com/transmit）和 Fetch（http://fetchsoftworks.com）。OS X 还有内置的 FTP 功能（参见 http://osxdaily.com/2011/02/07/ftp-from-mac/）。在网上搜索“FTP client”（FTP 客户端）可以找到更多的可以运行在 Windows 和 OS X 上的软件。它们的工作方式是相似的，只是有的软件功能多一些，有的少一些。

提 示　传输文件和文件夹时，它们是被复制到目标文件夹的，原来的位置还保留它们的原始版本。

提 示　如果目标位置已经存在要传输的文件或文件夹，那么 FTP 程序可能会弹出对话框，确认是否要覆盖（FileZilla 就会这样做）。不过，每个 FTP 客户端都不相同，因此有的软件也可能不经确认而直接覆盖。可以试着传一个测试文件，了解你的 FTP 客户端是如何处理这类情形的。

提 示　代码中的相对 URL 在文件夹传至服务器之后仍然有效。

提 示　如果访问网站的 URL 时网站无法显示，可能有多种原因。首先，再次检查是否已将文件传至正确的目录。通常，页面应位于 *public_html*、 *www* 或其他类似名称的目录下。你的 Web 主机提供商的操作说明中应该指明了正确的位置，如果不确定，可以向他们咨询。如果文件都放在正确的位置而网站依然无法显示，问题就可能出在域名服务器设置上（参见 21.2 节补充材料“连接域名和 Web 主机”）。

提 示　如果将某文件的新版本上传至服务器之后却没有看到修改的效果，可以清除浏览器的缓存，并再次检查页面。如果不清楚如何清除缓存，可以搜索浏览器的帮助文档。

附录 A

HTML 参考

本附录收录了几乎所有的 HTML 元素和属性，包括一些本书未作介绍的元素和属性。(大多数情况下，没有介绍它们是因为使用频率很低或者是高级特性。) 每个元素都有简单的描述，元素后面还列出了与其相关联的属性。

“参考章节”一列标出了本书介绍对应元素或属性的章节，方便查阅。有时，某个属性的章节可能对应的是对另一个元素的介绍，这意味着对该属性的介绍适用于所有可用的元素。

此外，有一些条目有以下标识：

- (5)，HTML5 新增的元素或属性；
- (*)，在 HTML5 之前的 HTML 中就已存在，但 HTML5 对其重新进行了定义。

HTML5 的一些新特性是旧的浏览器（如 IE8）所不支持的。关于最新的浏览器支持情况，参见 http://caniuse.com，该网站一直在不断更新中。

随时查阅

CSS 引用和 HTML 参考都可以在本书的配套网站上找到（见 www.htmlcssvqs.com）。

表 A.1　HTML 元素和属性

元素 / 属性	描　　述	参考章节
绝大多数 HTML 元素	下列属性适用于绝大多数 HTML 元素	
accesskey	用于为元素添加键盘快捷键	
aria-*	用于关联由 WAI-ARIA 指定的可访问性属性值	16.5
class	用于标识一组元素，以便为它们应用样式	3.14
contenteditable (5)	用于让元素的内容变成可编辑的	
data-* (5)	用于存储页面或应用特有的定制化数据	
dir	用于指定元素的文字方向	4.18
draggable (5)	用于让元素变成可拖拽的	
dropzone (5)	用于将元素标记为可拖拽元素可以放下的区域	
hidden (5)	用于指示元素还不是相关的或不再是相关的	
id	用于标识特定的元素，以便为其添加链接，应用样式或使用 JavaScript 编写脚本	3.14

（续）

元素 / 属性	描　述	参考章节
lang	用于指定元素的书写语言	3.1
role	用于为辅助设备提供关于元素角色的额外信息（由 WAI-ARIA 定义）	3.13
spellcheck (5)	用于指示是否应该对元素的内容进行拼写和语法检查	
style	用于添加本地样式表信息	8.4
tabindex	用于定义访问者使用制表键时在元素之间移动的顺序	6.1
title	用于为元素添加工具提示	3.15
a	用于创建链接和锚	6.1
href	用指于定链接到的页面的 URL 或锚的名称	6.1
hreflang (5)	用于指定所链接的资源的语言	
download (5)	用来指定指向下载资源的链接，将 download 值设置为文件名	
rel	用于标识链接的性质	6.1
target (*)	用于指定链接应打开的窗口或框架	6.1
type	用于指定资源的 MIME 类型	
abbr (*)	用于解释缩写或首字母缩写的含义	4.8
address	用于为最近的 article 或 body 元素祖先标识联系人信息	4.11
area	用于指定图像映射的坐标	
alt	用于给出关于区域的信息	
coords	用于给出图像映射中区域的坐标	
href	用于指定图像映射中区域链接的目标 URL	
hreflang (5)	用于指定所链接的资源的语言	
download (5)	用于指定指向下载资源的链接。应该将 download 值设置为文件名	
rel	用于标识链接的种类	
shape	用于指定图像映射中区域的形状	
target (*)	用于指定链接应打开的窗口或框架	6.1
article (5)	用于标识页面中的独立成分，原则上是可独立分发或可再用的	3.8
aside (5)	用于标识页面中的一个区域，其内容与周围的内容无关	3.10
audio (5)	用于在页面中嵌入音频	17.10
autoplay (5)	用于告诉浏览器在它能播放音频文件时立即开始播放	17.11
controls (5)	用于告诉浏览器为音频元素提供控件	17.10
loop (5)	用于告诉音频文件在播放到末尾后不间断地继续从头播放	17.11
muted (5)	用于控制音频输出的默认状态	
preload (5)	用于指定浏览器是否在访问者开始播放音频文件之前开始下载该文件	17.11
src (5)	用于标识要播放的音频文件的 URL	17.10
b (*)	用于标识出于实用的目的提醒读者注意的一块文本，不传达任何额外的重要性，也不表示其他的语态和语气	4.3
base	用于指定页面的基准 URL	
href	用于指定用于生成相对 URL 的 URL	

（续）

元素 / 属性	描　　述	参考章节
target (*)	用于指定页面上链接的默认目标	6.1
bdi (5)	用于标识独立于周围文本的用做双向文本格式化的一块文本	4.18
dir	用于指定文本方向	4.18
bdo (5)	用于显式地格式化其内容的文本方向	4.18
dir	用于指定文本方向	4.18
blockquote	用于指定页面上的一段引述文本	4.6
cite	用于给出来源的 URL	4.6
body	用于包围页面的主要内容区域	3.1
br	用于创建换行	4.16
button	用于创建按钮	16.15
autofocus (5)	用于指定按钮在页面加载时立即获得焦点	16.5
disabled	用于指示元素在当前状态下是不可用的	16.16
form (5)	用于将元素同另外一个不包含该元素的表单关联起来	
formaction (5)	用于覆盖表单的 action 属性	
formenctype (5)	用于覆盖表单的 enctype 属性	
formmethod (5)	用于覆盖表单的 method 属性	
formnovalidate (5)	用于覆盖表单的 novalidate 属性	16.15
formtarget (5)	用于覆盖表单的 target 属性	
name	用于标识使用按钮发送的数据，或者用于标识按钮本身（或许是为了使用某项 JavaScript 功能）	16.15
type	用于在表单元素中使用按钮	16.15
value	用于指定在点击按钮时应该提交的数据	16.15
canvas (5)	提供用于生成依赖于分辨率的位图画布的脚本，以在线呈现图像	17.14
width, height (5)	用于指定画布的尺寸	
caption	用于创建表格的标题	18.1
cite	用于将文本标记为引述	4.5
code	用于将文本标记为计算机代码	4.13
col	用于把表格中的列组合成非结构化组	
span	用于指定列组中的列数	
colgroup	用于把表格中的列组合成结构化列组	
span	用于指定列组中的列数	
datalist (5)	包含一组选项元素，这些元素是代表另一表单控件的一套预定义的选项	4.12
dd	用于标记列表中的定义	15.8
del	用于标记删除了的文本	4.12
cite	用于引用对修订进行解释的 URL	4.12
datetime	用于指定修订的时间和日期	4.7
details (5)	用于创建公开的小部件，访问者可以通过它获取额外的信息或控制	

（续）

元素 / 属性	描　述	参考章节
open (5)	用于指定元素在默认情况下是打开的还是关闭的	
dfn	用于指定列表项目的定义实例	4.9
title	用于提供术语的定义	4.9
div	用于将页面切割为块级区域	3.12
dl	用于创建描述列表	15.8
dt	用于标记要在列表中定义的术语	15.8
em (*)	用于标记要强调的文本	4.3
embed (*)	用于添加多媒体	
src	用于指定多媒体文件的 URL	
type	用于标识多媒体文件的 MIME 类型	
width, height	用于指定嵌入的多媒体播放器的尺寸	
fieldset	用于将一套表单元素组合在一起	16.4
disabled (5)	用于将表单元素组内的所有表单控件设为不可用的	16.16
form (5)	用于将元素同另外一个不包含该元素的表单关联起来	
figcaption (5)	用于为其父元素 figure 的内容提供标题或说明文字	4.4
figure (5)	用于识别在主文档流内被引用，但在不影响文档流的情况下可以移至他处的内容	4.4
footer (5)	用于识别最近的祖先元素 body、section、article 或 aside 的页脚	3.11
form	用于指定表单，表单用于收集要提交的数据	16.2
accept-charset	用于识别要在提交表单时使用的字符编码（默认为页面的字符集）	
action	用于给出处理表单数据的脚本的 URL	16.2
autocomplete (5)	当该属性设为 off 时，用于阻止浏览器提供或记住自动完成值（默认为 on，即在默认情况下允许自动完成）	16.5
enctype	用于确保文件以正确的编码格式发送至服务器	16.13
method	用于指定数据应如何发送至服务器	16.2
name	用于为表单提供名称，供以后使用	16.2
novalidate (5)	用于允许表单在不验证的情况下提交	16.2
target (*)	用于识别表单提交的目标窗口或 iframe	6.1
h1, h2, h3, h4, h5, h6	用于创建标题	3.3
head	用于创建 head 部分，该部分包含关于页面的信息，包括标题、制作者、关键字、样式表和脚本	3.1
header (5)	用于识别一组介绍性内容或导航帮助	3.5
hr (*)	用于标识段落级的主题变化	
html	用于标识作为 HTML 文档的文本文档	3.1
manifest (5)	用于指定离线时使用的应用程序缓存清单	
i (*)	用于标记用另外一种语态和语气，或不同于常规方式陈述以表现不同特质的一块文字	4.3
iframe	用于加载嵌套在另一个网页中的网页	
name	用于指定作为目标的 iframe 的名称	

（续）

元素 / 属性	描　述	参考章节
sandbox (5)	用于出于安全目的，为 iframe 的内容指定额外的限制	
seamless (5)	用于指定 iframe 是否显示为包含它的页面的一部分	
src	用于指定初始页面的 URL	
srcdoc (5)	用于指定初始页面的 URL	
width, height	用于指定 iframe 的尺寸	
img	用于在页面中插入图像	5.5
alt	用于提供替代文本。当图像无法显示时，则显示替代文本；替代文本也是为使用辅助设备的用户提供的	5.6
ismap	用于指示该元素可以提供对服务器端的图像映射（该元素必须是 a 元素的后代）的访问	
src	用于指定图像的 URL	5.5
usemap	用于指定应同引用图像一起使用的客户端图像映射	
width, height	用于指定图像的尺寸，从而让页面的加载更快，或出于对图像进行缩放的目的	5.7
input	用于创建表单元素	16.1
accept	当输入框类型为 file 时，用于向浏览器告知需要接受的文件类型	16.1
alt	当输入框类型为 image 时，用于提供替换文本	16.15
autocomplete (5)	当该属性设为 off 时，用于阻止浏览器提供或记住自动完成值（默认为 on，即在默认情况下允许自动完成）	16.5
autofocus (5)	用于指定输入框在页面加载时立即获得焦点	16.5
checked	用于标记单选按钮或复选框在默认情况下被选中	16.9
dirname (5)	用于标识输入的文本的方向	
disabled	用于指示输入框在当前状态下是不可用的	16.16
form (5)	用于将元素同另外一个不包含该元素的表单关联起来	
formaction (5)	用于覆盖表单的 action 属性	
formenctype (5)	用于覆盖表单的 enctype 属性	
formmethod (5)	用于覆盖表单的 method 属性	
formnovalidate (5)	用于覆盖表单的 novalidate 属性	16.15
formtarget (5)	用于覆盖表单的 target 属性	
list (5)	用于将输入框与数据列表关联起来	16.1
max, min (5)	用于指示输入框元素允许的值的范围	
maxlength	用于指定可输入到输入框元素的字符的最大数量	16.5
multiple (5)	用于指定是否允许用户输入一个以上的值	16.8
name	用于标识元素收集的数据	16.2
pattern (5)	用于提供可对输入框元素的数据进行检查的正则表达式	16.8
placeholder (5)	用于为数据输入提供提示	16.5
readonly	用于防止访问者修改特定的表单元素	16.14
required (5)	用于要求元素在提交表单时不能为空（当输入框类型为 hidden、image 或按钮类型时不可用）	16.5

（续）

元素 / 属性	描　述	参考章节
size	用于指定文本框或密码框的长度	16.5
src	用于指定活动图像的 URL	16.15
step (5)	用于控制允许输入的值的间隔大小和特殊性	
type	用于指定表单元素的类型为文本框、密码框、单选按钮、复选框、隐藏字段、提交按钮、重置按钮、活动图像、日期 / 时间框、数字框或颜色框；用于从一系列值中进行选择；或用于输入电话号码、电子邮件地址或一组搜索词	16.1
value	用于指定表单元素的默认数据	16.5
width, height	用于指定输入框的尺寸（仅在输入框类型为 image 时可以使用）	16.15
ins	用于标记对文档增加的内容	4.12
cite	用于引用对修订进行解释的 URL	4.12
datetime	用于指定修订的时间和日期	4.7
kbd	用于标记用户输入	4.13
keygen (5)	用于生成公钥—私钥对	
autofocus (5)	用于指定 keygen 元素在页面加载时立即获得焦点	16.5
challenge (5)	用于生成与密钥对伴生的诘问	
disabled (5)	用于指示元素在当前状态下是不可用的	16.16
form (5)	用于将元素同另外一个不包含该元素的表单关联起来	
keytype (5)	用于标识要生成的密钥对类型	
name (5)	用于标识收集的数据	16.2
label	用于为表单元素添加标签	16.6
for	用于指定标签所属的表单元素	16.6
form (5)	用于将元素同另外一个不包含该元素的表单关联起来	
legend	用于为表单元素组添加标签	16.4
li	用于创建列表项目	15.1
value	用于指定列表项目的初始值（当该元素为 ol 的子元素时）	15.4
link	用于指向外部样式表或其他外部资源	8.2
href	用于指定资源的 URL	8.2
hreflang (5)	用于指定所链接的资源的语言	
media	用于定义样式表的目标媒体类型和（或）媒体特性	8.6
rel	用于标识链接种类	8.2
sizes (5)	用于标识引用图标的大小（仅在 rel 属性为 icon 时可以使用）	
title	用于为替代样式表或其他资源添加标签	
type	用于指出资源的 MIME 类型（仅在链接类型不为 text/css 时需要使用）	
main	用于指定页面的主要内容区域	3.7
map (5)	用于创建客户端图像映射	
name	用于对映射命名，从而使其可在以后被引用	
mark (5)	出于引用的目的，对与另一个上下文相关的文本进行突出显示	4.15

（续）

元素 / 属性	描　述	参考章节
menu (*)	用于包含命令列表	
label (5)	用于为菜单添加标签	
type (5)	用于标识所使用的菜单的种类：context、list（默认值）或 toolbar	
meta	用于关联页面的各种元数据	3.1
charset	用于标识页面本身的字符编码	3.1
content	用于添加关于页面本身的额外信息	
http-equiv	用于创建指向其他页面的自动跳转，设置默认脚本语言，声明字符编码	
name	用于标识关于页面的额外信息	
meter (5)	用于表示在已知范围内的量度	4.18
high, low (5)	用于指定量度为 high 或 low	4.18
max, min (5)	用于标识允许指定的值的最大值和最小值	4.18
name (5)	用于标识收集的数据	4.18
optimum (5)	用于标识最优值	4.18
value (5)	用于指定量表的当前值（必需的属性）	4.18
nav (5)	用于标识页面的一块区域，该区域包含指向其他页面或页面内不同部分的链接	3.6
noscript	用于提供脚本的替代内容	
object	用于在网页中嵌入对象	
data	用于标识要嵌入的多媒体文件的来源	
form (5)	用于将元素同另外一个不包含该元素的表单关联起来	
name	用于标识对象（例如，对其编写脚本）	
type	用于指出对象的 MIME 类型	
typemustmatch	用于指示对象 data 属性中指定的资源的 MIME 类型必须与对象 type 属性中标识的 MIME 类型相同	
usemap	用于指示对象是否拥有相关联的图像映射	
width, height	用于指定对象框的尺寸	
ol	用于创建有序列表	15.1
reversed (5)	用于指定列表是否为反序（…，3，2，1）	15.1
start (*)	用于指定第一个列表项目的初始值	15.4
type (*)	用于指定每个列表项目开始的数字类型	15.2
optgroup	用于对 select 元素中的 option 元素进行分组，一组内的 option 元素位于同一个标签下	16.12
disabled	用于指示元素在当前状态下是不可用的	16.16
label	用于为选项组添加标签	16.12
option	用于创建 select 或 datalist 元素中的单独的选项	16.12
disabled	用于指示元素在当前状态下是不可用的	16.16
label	用于指定选项如何出现在菜单中	16.12
selected	用于标记空白表单中默认被选中的菜单选项	16.12

（续）

元素 / 属性	描　　述	参考章节
value	用于指定菜单选项的初始值	16.12
output (5)	用于表示计算结果	16.1
for (5)	用于创建计算结果与进入计算过程的值之间的显式关联	
form (5)	用于将元素同另外一个不包含该元素的表单关联起来	
name (5)	用于标识收集的数据	16.2
p	用于创建段落	4.1
param	用于设置对象的属性	
name	用于标识属性的种类	
value	用于设置有名称的属性的值	
pre	用于表示一块预格式化文本	4.14
progress (5)	用于标识任务的完成进度	4.18
max (5)	必须为大于 0 的有效浮点数（如果有的话）	4.18
value (5)	必须为大于或等于 0 的有效浮点数（且小于或等于 max 属性的值，如果有该值的话）	4.18
q	用于引用来自另一来源的短文	4.6
cite	用于给出引用源的 URL	4.6
rp (5)	用于在不支持旁注标记的浏览器中的旁注标记文本周围显示括号	4.18
rt (5)	用于标记旁注标记文本	4.18
ruby (5)	用于允许文本被旁注标记所标记	4.18
s (*)	用于标识不再准确或不再相关的文本	4.12
samp	用于呈现某程序或计算系统的样本输出	4.13
script	用于为页面添加“自动的”脚本	19.1
async (5)	用于影响脚本的加载和执行	
charset	用于指定外部脚本所用的字符集	3.1
defer	用于影响脚本的加载和执行	
src	用于引用外部脚本	19.1
type (*)	用于指定脚本所用的脚本语言（仅在脚本类型不为 text/javascript 时需要使用）	
section (5)	用于识别文档的区块	3.9
select	用于创建可以从一组选项中进行选择的表单控件	16.12
autofocus (5)	用于指定 select 元素在页面加载时立即获得焦点	16.5
disabled	用于指示元素在当前状态下是不可用的	16.16
form (5)	用于将元素同另外一个不包含该元素的表单关联起来	
multiple	用于允许用户在菜单中选择一个以上的选择	16.12
name	用于标识从菜单收集的数据	16.12
required (5)	用于标识用户必须选择一个选项才能提交表单（第一个 option 子元素必须为占位符或空值）	16.5
size	用于指定在初始状态下菜单中可见的项目数（同时用于将菜单显示为列表）	16.12

（续）

元素 / 属性	描　述	参考章节
small (*)	用于呈现像条文细则等次要注释	4.2
source (5)	用于在 audio 或 video 元素中标识替代媒体资源	17.12 17.7
media (5)	用于标识资源的目标媒体类型	12.4
src (5)	用于标识要播放的音频或视频文件的 URL	17.7
type (5)	用于指出资源的 MIME 类型	17.7
span	用于包围元素中无直接语义含义的内容	4.17
strong (*)	用于标识元素内特别重要的内容	4.3
style	用于在页面中嵌入样式信息	8.3
media	用于标识样式表的用处	12.4
title	为其他样式表标记说明标签	12.4
type (*)	用于标识样式表的 MIME 类型（仅在样式类型不为 text/css 时需要使用）	
sub	用于创建下标	4.10
summary (5)	用于标识 details 父元素内容的摘要、标题或说明文字	
sup	用于创建上标	4.10
svg (5)	用于在页面中嵌入可缩放矢量图形	17.14
table	用于创建表格	18.1
tbody	用于识别表格的主体部分；比之于头部（thead）和尾部（tfoot）	18.1
td, th	分别用于在表格中创建普通单元格和标题单元格	18.1
colspan	用于让单元格跨越多列	18.2
headers	通过在 headers 值中包含某个 th 的 id 值，从而将这个 th 与一个 td 或者另一个 th 显式关联	
rowspan	用于让单元格跨越多行	18.2
scope	用于指定 th 应用于哪些行、列、行组或列组	18.1
textarea	用于在表单中创建文本块输入区域	16.11
autofocus (5)	用于指定文本区域元素在页面加载时立即获得焦点	16.5
dirname (5)	用于识别输入的文本的方向	
disabled	用于指示元素在当前状态下是不可用的	16.16
form (5)	用于将元素同另外一个不包含该元素的表单关联起来	
maxlength	用于指定可输入到 textarea 元素的字符的最大数量	16.11
name	用于标识从文本块收集的数据	16.11
placeholder (5)	用于为数据输入提供提示	16.5
readonly	用于保护文本区域的内容	16.14
required (5)	用于要求元素在提交表单时不能为空	16.5
rows, cols	分别用于指定文本块的行数和列数	16.11
wrap (5)	用于指定在字段内容提交时使用软换行或硬换行	
tfoot, thead	用于识别表格的尾部和头部	18.1

（续）

元素 / 属性	描　　述	参考章节
time (5)	用于指定日期和时间	4.7
datetime (5)	用于为元素的文本所表达的时间或日期提供机器可读版本	4.7
title	用于创建页面的标题（必须使用）	3.2
tr	用于在表格中创建行	18.1
track (5)	用于为 audio 或 video 父元素指定外部计时文本轨道	17.8
default (5)	用于指示默认轨道	
kind (5)	用于识别轨道为 subtitles（字幕）、captions（标题）、descriptions（描述）、chapters（篇章）或 metadata（元数据）	
label (5)	用于为轨道提供用户可读的名称	
src (5)	用于标识轨道数据的 URL	
srclang (5)	用于标识轨道数据的语言	
u (*)	用于显示一段文本，作为虽然明确地呈现却不怎么准确的非文本注解	4.18
ul	用于创建无序列表	15.1
var	用于标记作为变量的文本	4.13
video (5)	用于嵌入视频、电影和有说明文字的音频文件	17.3
autoplay (5)	用于告诉浏览器在它能播放视频文件时立即开始播放	17.4
controls (5)	用于告诉浏览器为视频元素提供控件	17.4
loop (5)	用于告诉视频文件在播放到末尾后不间断地继续从头播放	17.5
muted (5)	用于控制音频输出的默认状态	
poster (5)	用于指定占位图像的 URL，该图像在媒体加载时或加载出现问题时显示	17.5
preload (5)	用于指定浏览器是否在访问者开始播放媒体文件之前开始下载该文件	17.6
src (5)	用于标识要播放的视频文件的 URL	17.3
width, height (5)	用于指定视频的尺寸	17.3
wbr (5)	用于识别在没有连字符的单词中可以在必要时进行换行的位置	4.18

附录 B

CSS 引用

B

本附录包含以下 CSS 引用表格。

- 表 B.1　CSS 属性和值，可用做很多常见或实用的 CSS 属性及其默认值、允许值的快速参考。
- 表 B.2　CSS 选择器和结合符，对 CSS 选择器和结合符的引用，包括在 CSS3 中引入的那些。
- 表 B.3　CSS3 的颜色值，涵盖了在 CSS3 中引入的颜色值（HSL、HSLA 和 RGBA）。
- 表 B.4　媒体查询。

关于哪些浏览器支持哪些 CSS3 属性和值的信息，可参考 www.quirksmode.org/css/contents.html、http://caniuse.com 及 http://findmebyip.com/litmus。此外，还可以使用像 Modernizr（www.modernizr.com）这样的 JavaScript 库测试浏览器对这些特性的支持情况。

B.1　CSS 属性和值

表 B.1　CSS 属性和值

属性 / 值	描述和注释
background 任　何 background-attachment、background-color、background-image、background-repeat 和（或）background-position 值的组合，或 inherit	用于修改元素的背景颜色和背景图像 初始值取决于单独的属性，不继承的 background-position 可使用百分数 如果要显示多重背景，可使用逗号分隔组合背景值；如果要指定 background-color，应包含在最后一个背景中
background-attachment scroll、fixed 或 inherit	用于决定背景图像是否可以滚动，以及滚动的方式 初始值：scroll；不继承的 如果要显示多重背景，可以为每个背景应用不同的 background-attachment 值（用逗号分隔不同的值）
background-color 颜色值、transparent 或 inherit	用于设置元素的背景颜色 初始值：transparent；不继承的
background-image URL、CSS 渐变、none 或 inherit	用于设置元素的背景图像 初始值：none；不继承的 如果要显示多重背景，可使用逗号分隔图像值

（续）

属性 / 值	描述和注释
background-position 一个或两个百分数、长度（或一个百分数和一个长度），或 top、center、bottom 之一和（或）left、center、right 之一，或使用 inherit	用于设置指定的背景图像的物理位置 初始值：0% 0%；如果仅设置一个百分数，它会用于水平位置，而垂直位置的初始值则被设为 50%；如果仅使用一个关键字，另一个的初始值则为 center；应用于块级和替换元素；不继承的；百分数相对于盒本身的尺寸 如果要显示多重背景，可以为每个背景应用不同的 background-position 值（用逗号分隔不同的值）
background-repeat repeat、repeat-x、repeat-y、no-repeat 或 inherit 之一	用于确定背景图像是否重复及重复方式 初始值：repeat；不继承的 如果要显示多重背景图像，可以为每个背景图像应用不同的 background-repeat 值（用逗号分隔不同的值）
background-size 一个或两个百分数或长度，或 auto，或使用 cover 或 contain	用于指定背景图像的尺寸 初始值：auto；不继承的 如果要显示多重背景图像，可以为每个背景图像应用不同的 background-size 值（用逗号分隔不同的值）
border 任何 border-width、border-style 值和（或）颜色值的组合，或 inherit	用于定义元素四边边框的所有要素 初始值取决于单独的属性；不继承的
border-color 一至四个颜色值、transparent 或 inherit	用于指定元素的一个或多个边的边框的颜色 初始值：元素的 color 属性；不继承的
border-radius	用于为盒创建圆角 初始值：0；不继承的
border-top-right-radius border-bottom-right-radius border-bottom-left-radius border-top-left-radius	用于为盒的一个角设置 border-radius 值 初始值：0；不继承的 注：Firefox 的旧版本使用与此不同的语法创建单独的圆角：-moz-border-radius-topright、-moz-border-radius-bottomright、-moz-border-radius-bottomleft、-moz-border-radius-topleft
border-spacing 一个或两个长度，或 inherit	用于指定表格边框之间的空隙大小 初始值：0；仅可应用于表格元素；继承的
border-style 一至四个以下的值：none、dotted、dashed、solid、double、groove、ridge、inset、outset、inherit	用于为元素的一个或多个边设置边框样式 初始值：none；不继承的
border-top、border-right、border-bottom、border-left 任何用于 border-width、border-style 的单个值和（或）颜色值的组合，或使用 inherit	用于为元素的一个边一次性定义全部三个边框属性 初始值取决于单独的值；不继承的
border-top-color、border-right-color、border-bottom-color、border-left-color 颜色值或 inherit	用于为元素的一个边定义边框颜色 初始值：color 属性的值；不继承的
border-top-style、border-right-style、border-bottom-style、border-left-style none、dotted、dashed、solid、double、groove、ridge、inset、outset 或 inherit	用于为元素的一个边定义边框样式 初始值：none；不继承的

（续）

属性 / 值	描述和注释
border-top-width、border-right-width、border-bottom-width、border-left-width thin、medium、thick 或长度	用于为元素的一个边定义边框宽度 初始值：mdeium；不继承的
border-width 一至四个以下的值：thin、medium、thick 或长度	用于为元素的一个或四个边定义边框宽度 初始值：mdeium；不继承的
bottom 百分数、长度、auto 或 inherit	用于设置元素相对于其父元素底部边缘的位移大小 初始值：auto；不继承的；百分数相对于包含块的高度
box-shadow 可选的 inset，接着是二至四个长度值，接着是颜色值	用于为盒添加一个或多个阴影。长度值（依次）表示相对于盒右侧的位置（负数则表示相对于盒左侧的位置）、相对于盒底部的位置（负数则表示相对于盒顶部的位置）、模糊半径（不可为负数）和伸展距离（负数会让阴影收缩）。每个 box-shadow 值之间用逗号分隔 初始值：none；继承的
clear none、left、right、both 或 inherit	用于防止元素包围在浮动元素的一边或两边 初始值：none；只能应用于块级元素；不继承的
clip auto、rect 或 inherit	用于仅显示元素的一部分 初始值：auto；只能应用于绝对定位的元素
color 颜色值或 inherit	用于设置元素的文本颜色 初始值：父元素的颜色；有的颜色是由浏览器设置的；继承的
cursor auto、crosshair、default、pointer、progress、move、e-resize、ne-resize、nw-resize、n-resize、se-resize、sw-resize、s-resize、w-resize、text、wait、help、URL 或 inherit 之一	用于设置指针的形状 初始值：auto；继承的
display inline、block、inline-block、list-item、run-in、compact、table、inline-table、table-row-group、table-header-group、table-footer-group、table-row、table-column-group、table-column、table-cell、table-caption、ruby、ruby-base、ruby-text、ruby-base-group、ruby-text-group、none、inherit 之一	用于确定元素如何显示，以及是否显示 初始值：通常为 inline 或 block；不继承的
float left、right、none、inherit 之一	用于确定元素向父元素的哪一边浮动 初始值：none；不可应用于定位过的元素[①]或生成的内容；不继承的
font 如果需要，任何 font-style、font-variant 和 font-weight 值的组合，接着是必需的 font-size、可选的 line-height 值和必需的 font-family，或使用 inherit	用于设置文本的字体系列、字体大小（这二者是必需的）及可选的字体样式、变体、粗细和行高 初始值取决于单独的属性；继承的；font-size 和 line-height 可使用百分数；font-size 和 font-family 是必需的，否则 font 属性是无效的

① 即设为绝对定位、相对定位或固定定位的元素。——译者注

（续）

属性 / 值	描述和注释
font-family 一个或多个由引号包着的字体名称，接着是可选的表示类属的字体名称，或使用 inherit	用于为文本选择字体系列 初始值：取决于浏览器；继承的
font-size 绝对大小、相对大小、长度、百分数或 inherit	用于设置文本的大小 初始值：medium；计算的值是继承的；百分数相对于父元素的字体大小
font-style normal、italic、oblique 或 inherit	用于将文本标记为斜体 初始值：normal；继承的
font-variant normal、small-caps 或 inherit	用于设置小型大写字母 初始值：normal；继承的
font-weight normal、bold、bolder、lighter、100、200、300、400、500、600、700、800、900 或 inherit	用于应用、移除、调整粗体格式 初始值：normal；数字值当做关键字而非整数进行处理（例如，不能使用 150）；继承的
height 长度、百分数、auto 或 inherit	用于设置元素的高度 初始值：auto；可应用于除了非替换行内元素、表格列和列组以外的任何元素；不继承的
left 长度、百分数、auto 或 inherit	用于设置元素相对于其父元素左侧边缘的位移大小 初始值：auto；只能用于定位过的元素；不继承的；百分数相对于包含块的宽度
letter-spacing normal、长度或 inherit	用于设置字母之间的间隙大小 初始值：normal；继承的
line-height normal、数字、长度、百分数或 inherit	用于设置文本行之间的距离 初始值：normal；继承的；百分数相对于元素自身的字体大小
list-style 任何 list-style-type、list-style-position 和（或）list-style-image 值的组合，或使用 inherit	用于设置列表的标识（常规的或定制的）及其位置 初始值取决于单独元素的初始值；只能应用于列表元素；继承的
list-style-image URL、none 或 inherit	用于为列表指定定制的标识 初始值：none；只能应用于列表元素；覆盖 list-style-type；继承的
list-style-position inside、outside 或 inherit	用于确定列表标识的位置 初始值：outside；只能应用于列表元素；继承的
list-style-type disc、circle、square、decimal、lower-roman、upper-roman、lower-alpha、upper-alpha、none 或 inherit	用于设置列表的标识 初始值：disc；只能应用于列表元素；如果 list-style-type 是有效的则不使用；继承的
margin 一至四个以下的值：长度、百分数、auto 或 inherit	用于设置元素与其父元素和（或）同胞元素之间在一个或多个边上的间隔大小 初始值取决于浏览器和 width 值；不继承的；百分数相对于包含块的宽度

（续）

属性 / 值	描述和注释
margin-top、margin-right、margin-bottom、margin-left 长度、百分数、auto 或 inherit	用于设置元素与其父元素和（或）同胞元素之间在一个边上的间隔大小 初始值：0；不继承的；百分数相对于包含块的宽度；如果 width、margin-right 和 margin-left 之和大于父元素的包含块，则 margin-right 和 margin-left 的值会被覆盖
max-height、max-width 长度、百分数、none 或 inherit	分别用于设置元素的最大高度和（或）最大宽度 初始值：none；不能用于行内元素或表格元素；不继承的；百分数相对于包含块的高度 / 宽度
min-height、max-width 长度、百分数或 inherit	分别用于设置元素的最小高度和（或）最小宽度 初始值：none；不能用于行内元素或表格元素；不继承的；百分数相对于包含块的高度 / 宽度
opacity 0.0（表示完全透明）至 1.0（表示完全不透明）之间的任何小数	用于让元素半透明或不可见 初始值：1；不继承的
orphans 整数或 inherit	用于指定元素可以单独出现在页面底部的行数 初始值：2；只能用于块级元素；继承的；仅用于打印媒体
overflow visible、hidden、scroll、auto 或 inherit	用于确定当内容超出元素内容区域时额外的内容如何显示 初始值：visible；只能用于块级元素和替换元素；不继承的
padding 一至四个长度或百分数，或使用 inherit	用于指定元素内容区域和边框之间在一个或多个边上的距离 初始值取决于浏览器；不继承的；百分数相对于包含块的宽度
padding-top、padding-right、padding-bottom、padding-left 长度、百分数或 inherit	用于指定元素内容区域和边框之间在一个边上的距离 初始值：0；不继承的；百分数相对于包含块的宽度
page-break-after、page-break-before always、avoid、auto、right、left 或 inherit	用于指定什么时候应出现分页，什么时候不应出现 初始值：auto；只能用于块级元素；不继承的；仅用于打印媒体
page-break-inside avoid、auto 或 inherit	阻止跨页的元素产生分页 初始值：auto；只能用于块级元素；继承的；仅用于打印媒体
position static、relative、absolute、fixed 或 inherit	用于确定元素如何相对于文档流进行定位 初始值：static；不继承的
right 长度、百分数、auto 或 inherit	用于设置元素相对于其父元素右侧边缘的位移大小 初始值：auto；只能用于定位过的元素；不继承的；百分数相对于包含块的宽度
table-layout fixed、auto 或 inherit	用于选择确定单元格宽度的算法 初始值：auto；不继承的
text-align left、right、center、justify、字符串或 inherit	用于指定文本对齐方式 初始值取决于浏览器和书写方向；只能应用于块级元素；继承的
text-decoration 任何 underline、overline、line-through 和 blink 的组合，或 none、inherit	用于修饰文本（大多数为线条） 初始值：none；不继承的

（续）

属性 / 值	描述和注释
text-indent 长度、百分数或 inherit	用于设置段落第一行的缩进量 初始值：0；只能应用于块级元素；继承的；百分数相对于包含块的宽度
text-overflow clip、ellipsis 或 "string"	用于指定文本不可见时处理溢出的方式 初始值：clip
text-shadow 两个或四个长度值，接着是颜色值	用于为元素的文本添加一个或多个阴影。长度值（依次）表示相对于文本右侧的位置（负数则表示相对于文本左侧的位置）、相对于文本底部的位置（负数则表示相对于文本顶部的位置）、模糊半径（不可为负数）和伸展距离（负数会让阴影收缩）。每个 text-shadow 值之间用逗号分隔 初始值：none；继承的
text-transform capitalize、uppercase、lowercase、none 或 inherit	用于设置元素的文本的大小写 初始值：none；继承的
transform none 或一系列变形功能（matrix、translate、translateX、translateY、scale、scaleX、scaleY、rotate、skew、skewX、skewY）	用于对元素进行形状、大小或方向上的变形 初始值：none；不继承的；变形功能按照它们所列的顺序进行应用
transform-origin 一个或两个百分数或长度（或一个百分数和一个长度），或 top、center、bottom 之一和（或）left、center、right 之一	用于定义应用于元素的变形的起点 初始值：50% 50%；不继承的；只能应用于块级元素和行内元素；百分数相对于元素盒的大小
transition 依次定义 transition-property、transition-duration、transition-timing-function 和 transition-delay 的简记法（用空格分隔）	用于为元素定义变形效果 初始值取决于单独的属性；可应用于所有的元素，包括 :before 和 :after 伪元素；值的顺序对此属性很重要
transition-property none、all 或用逗号分隔的一组 CSS 属性	用于识别在应用了变形的元素上定义的 CSS 属性 初始值：all；不继承的；可应用于所有的元素，包括 :before 和 :after 伪元素
transition-duration 以秒或毫秒为单位的时间值	用于确定完成变形所需的时间 初始值：0s（0 秒）；不继承的；可应用于所有的元素，包括 :before 和 :after 伪元素
transition-timing-function ease、linear、ease-in、ease-out、ease-in-out、cubic-bezier(number, number, number, number)	描述用于变形计算过程的中间值的使用方法 初始值：ease；可应用于所有的元素，包括 :before 和 :after 伪元素
transition-delay 以秒或毫秒为单位的时间值	用于定义变形开始的时间 初始值：0s（0 秒）；不继承的；可应用于所有的元素，包括 :before 和 :after 伪元素
top 长度、百分数、auto 或 inherit	用于设置元素相对于其父元素顶部边缘的位移大小 初始值：auto；只能用于定位过的元素；不继承的；百分数相对于包含块的高度

（续）

属性 / 值	描述和注释
vertical-align baseline、sub、super、top、text-top、middle、bottom、text-bottom、百分数、长度或 inherit	用于指定元素在垂直方向上的对齐方式 初始值：baseline；不能应用于行内元素和表格单元格元素；不继承的；百分数相对于元素的 line-height 属性
visibility visible、hidden、collapse 或 inherit	用于在不将元素移出文档流的情况下让元素不可见 初始值：inherit，事实上是不继承的（仍存争议）
white-space normal、pre、nowrap、pre-wrap、pre-lined 或 inherit	用于指定如何处理空格 初始值：normal；只能用于块级元素；继承的
widows 整数或 inherit	用于指定元素可以单独出现在页面顶部的行数 初始值：2；只能用于块级元素；继承的；仅用于打印媒体
width 长度、百分数、auto 或 inherit	用于设置元素的宽度 初始值：auto；不能应用于行内元素、表格行或行组；不继承的；百分数相对于包含块的宽度
word-spacing normal、长度或 inherit	用于设置单词之间的距离 初始值：normal；继承的
z-index auto、整数或 inherit	用于设置元素相对于重叠元素的深度 初始值：auto；只能应用于定位了的元素；不继承的

表 B.1 是根据 www.w3.org/TR/CSS21/propidx.html 提供的完整规范制订的，版权由万维网联盟（美国麻省理工学院、法国国家计算机科学与控制研究所、日本庆应义塾大学）所有。保留所有权利。

B.2　CSS 选择器和结合符

表 B.2　CSS 选择器和结合符

模　式	含　义	CSS3	选择器类型
*	任何元素		通用选择器
E	类型为 E 的元素		类型选择器
E[foo]	带 "foo" 属性的 E 元素		属性选择器
E[foo="bar"]	"foo" 属性值恰为 "bar" 的 E 元素（引号是可选的）		属性选择器
E[foo ~ ="bar"]	"foo" 属性为一组空格分隔的值，且其中之一恰为 "bar" 的 E 元素（引号是可选的）		属性选择器
E[foo^="bar"]	"foo" 属性以 "bar" 开头的 E 元素（引号是可选的）	是	属性选择器
E[foo$="bar"]	"foo" 属性以 "bar" 结束的 E 元素（引号是可选的）	是	属性选择器

模　　式	含　　义	CSS3	选择器类型
E[foo*="bar"]	"foo" 属性值在某处包含 "bar" 的 E 元素（引号是可选的）	是	属性选择器
E[foo|="en"]	"foo" 属性为一组连字符分隔的值且（从左边开始）以 "en" 开头的 E 元素（引号是可选的）		属性选择器
E:root	E 元素，文档根元素	是	结构伪类
E:nth-child(n)	E 元素，其父元素的第 n 个子元素	是	结构伪类
E:nth-last-child(n)	E 元素，其父元素的倒数第 n 个子元素	是	结构伪类
E:nth-of-type(n)	E 元素，该类型的第 n 个同胞元素	是	结构伪类
E:nth-last-of-type(n)	E 元素，该类型的倒数第 n 个同胞元素	是	结构伪类
E:first-child	E 元素，其父元素的第一个子元素		结构伪类
E:last-child	E 元素，其父元素的最后一个子元素	是	结构伪类
E:first-of-type	E 元素，该类型的第一个同胞元素	是	结构伪类
E:last-of-type	E 元素，该类型的最后一个同胞元素	是	结构伪类
E:only-child	E 元素，其父元素的唯一子元素	是	结构伪类
E:only-of-type	E 元素，该类型的唯一同胞元素	是	结构伪类
E:empty	没有子元素（含文本结点）的 E 元素	是	结构伪类
E:link, E:visited	作为目标尚未访问过（:link）或已经访问过（:visited）的超链接的 E 元素		链接伪类
E:focus, E:hover, E:active	处于特定用户操作下的 E 元素		用户操作伪类
E:target	作为引用 URI 目标的 E 元素	是	目标伪类
E:lang(fr)	语言为 "fr" 的 E 元素		:lang() 伪类
E:enabled E:disa-bled	状态为有效的或无效的用户界面 E 元素	是	UI 元素状态伪类
E:checked	选中了的用户界面 E 元素（如单选按钮或复选框）	是	UI 元素状态伪类
E::first-line	E 元素在格式上的第一行		::first-line 伪元素
E::first-letter	E 元素在格式上的第一个字母		::first-letter 伪元素
E::before	E 元素之前的生成内容		::before 伪元素
E::after	E 元素之后的生成内容		::after 伪元素
E.warning	类为 "warning" 的 E 元素		类选择器
E#myid	ID 等于 "myid" 的 E 元素		ID 选择器
E:not(s)	与简单选择器 s（例如 input:not(.warning)）不匹配的 E 元素	是	否定伪类

（续）

模　　式	含　　义	CSS3	选择器类型
E F	作为 E 元素后代的 F 元素		后代结合符
E > F	作为 E 元素子元素的 F 元素		子元素结合符
E + F	紧接 E 元素后面的 F 元素		相邻同胞元素结合符
E ~ F	位于 E 元素后面的 F 元素	是	通用同胞元素结合符

B.3　CSS3 颜色值

表 B.3　CSS3 颜色值

颜色值	描述和注释
rgb(red-value, green-value, blue-value)	RGB（红、绿、蓝）颜色模式 值可以是 0 到 255 之间的数字或百分数（不能是数字和百分数的组合） rgb(0, 0, 0) 和 rgb(0%, 0%, 0%) 为黑色 rgb(255, 255, 255) 和 rgb(100%, 100%, 100%) 为白色
rgba(red-value, green-value, blue-value, alpha)	RGB 颜色模式，加上 alpha 透明度 颜色值同 RGB 语法相同 第四个参数 alpha 是大于等于 0.0（完全透明）且小于等于 1.0（完全不透明）的小数
hsl(hue-value, saturation-value, lightness-value)	HSL（色相、饱和度、亮度）颜色模式 色相值用颜色环的角度（0 至 360 之间的数字）表示：0 和 360 为红色，120 为绿色，240 为蓝色，位于之间的其他值表示其他颜色 饱和度值用百分数表示：0% 为灰色，100% 为完全饱和的颜色 亮度值用百分数表示：0% 为黑色，100% 为白色，50% 为“正常”
hsla(hue-value, saturation-value, lightness-value, alpha)	HSL 颜色模式，加上 alpha 透明度 颜色值同 HSL 语法相同 第四个参数 alpha 是大于等于 0.0（完全透明）且小于等于 1.0（完全不透明）的小数

B.4　媒体查询

表 B.4　媒体查询

特　　性	描述和注释
width min-width max-width 长度	输出设备的目标显示区域的宽度、最小宽度或最大宽度 应用：可视媒体和触觉媒体
height min-height max-height 长度	输出设备的目标显示区域的高度、最小高度或最大高度 应用：可视媒体和触觉媒体

（续）

特　　性	描述和注释
device-width min-device-width max-device-width 长度	输出设备的呈现表面的宽度、最小宽度或最大宽度 应用：可视媒体和触觉媒体
device-height min-device-height max-device-height 长度	输出设备的呈现表面的高度、最小高度或最大高度 应用：可视媒体和触觉媒体
orientation portrait 或 landscape	当 height 特性值大于或等于 width 特性值时，方向为 portrait；否则为 landscape 应用：位图媒体
aspect-ratio min-aspect-ratio max-aspect-ratio 比例（如 4/3 或 16/9）	width 特性值与 height 特性值的比例、最小比例或最大比例 应用：位图媒体
device-aspect-ratio min-device-aspect-ratio max-device-aspect-ratio 比例（如 4/3 或 16/9）	device-width 特性值与 device-height 特性值的比例、最小比例或最大比例 应用：位图媒体
color min-color max-color 整数	输出设备每种颜色的位数、最小位数或最大位数；如果设备不是彩色的设备，则值为 0 应用：可视媒体
color-index min-color-index max-color-index 整数	输出设备颜色查询表中项目的数量、最小数量或最大数量；如果设备不使用颜色查询表，则值为 0 应用：可视媒体
monochrome、 min-monochrome、 max-monochrome 整数	在单色帧缓冲中每像素的位数、最小位数或最大位数；如果设备不是单色的设备，则值为 0 应用：可视媒体
resolution min-resolution max-resolution 分辨率（如 300dpi 或 118dpcm）	输出设备的分辨率、最小分辨率或最大分辨率（即像素密度）；resolution（不是 min-resolution 或 max-resolution）不会检测使用非方形像素的设备
scan progressive 或 interlace	电视输出设备的扫描过程 应用：电视媒体
grid 0 或 1	设备是基于栅格的还是基于位图的；如果输出设备是基于栅格的（如 TTY 终端）则值为 1，否则值为 0；这种媒体查询也可以使用不带值的形式表示（例如 @media grid） 应用：可视媒体和触觉媒体

（第8版）

HTML5与CSS3基础教程

"我们这些有过不少实际经验的设计师往往想当然地认为自己什么都知道，事实并非如此。本书告诉我们，我们知道的很多东西其实都是错的。所有Web设计师都需要看看这本书。"

——Web标准计划创始人Jeffrey Zeldman
对本书第6版的评论

"这本书让我读起来很踏实。"

——豆瓣读者对本书第7版的评论

"自从2006年购买第5版，我就认准这本书了。第8版不论从内容组织还是示例讲解上都有不少惊喜，从中可以看出作者在新版本上耗费了大量心血！"

——亚马逊读者评论

"本书介绍如何设计、组织以及格式化网站，是Web开发方面的经典教程。"

——hostucan.net

本书是风靡全球的HTML和CSS入门教程的最新版，至第6版累积销量已超过100万册，被翻译为十多种语言，长期雄踞亚马逊书店计算机图书排行榜榜首。

第8版秉承作者直观透彻、循序渐进、基础知识与案例实践紧密结合的讲授特色，采用独特的双栏图文并排方式，手把手指导读者从零开始轻松入门。相较第7版，全书2/3以上的内容进行了更新，全面反映了HTML5和CSS3的最新特色，细致阐述了响应式Web设计与移动开发等热点问题。书中主要内容包括：如何创建HTML5页面，如何使用HTML5元素，如何用CSS3为网页添加样式，如何向页面添加JavaScript代码，如何测试做好的页面并将其上传到万维网。另外，本书强调渐进增强这种网站设计方法的重要性，并将其贯穿在全书的具体实践中。

作者专为本书设计了内容丰富的配套网站htmlcssvqs.com，提供海量精彩示例、HTML与CSS元素及属性列表以及其他附加材料，方便读者随时参考与引用。

Peachpit Press

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机/Web开发

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-35065-7

ISBN 978-7-115-35065-7
定价：69.00元

欢迎加入

图灵社区

电子书发售平台

电子出版的时代已经来临，在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版的梦想。你可以联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者，这极大地降低了出版的门槛。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果有意翻译哪本图书，欢迎来社区申请。只要通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

读者交流平台

在图灵社区，读者可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。欢迎大家积极参与社区开展的访谈、审读、评选等多种活动，赢取银子，可以换书哦！